

**Ein Protokollentwurfs- und -spezifikationsverfahren
auf Basis einer erweiterten Zustandsdiagrammtechnik
in Listenform mit Vorverarbeitung von Ereignissen**

DISSERTATION

zur Erlangung des Grades eines
Doktor-Ingenieurs

der Abteilung für Elektrotechnik
an der Ruhr-Universität Bochum

von

REINHOLD HÖLZGEN

Dernbach/Westerwald

Bochum 1988



Ein Protokollentwurfs- und -spezifikationsverfahren
auf Basis einer erweiterten Zustandsdiagrammtechnik
in Listenform mit Vorverarbeitung von Ereignissen

DISSERTATION

Digitized by the Internet Archive
in 2026

Zur Erlangung des Doktorgrades
an der Ruhr-Universität Bochum

WERNER HÖTZGEN

Deutsch/Westfälisch

Bochum 1986

**Ein Protokollentwurfs- und -spezifikationsverfahren
auf Basis einer erweiterten Zustandsdiagrammtechnik
in Listenform mit Vorverarbeitung von Ereignissen**

DISSERTATION

zur Erlangung des Grades eines
Doktor-Ingenieurs

der Abteilung für Elektrotechnik
an der Ruhr-Universität Bochum

von

REINHOLD HÖLZGEN

Dernbach/Westerwald

Bochum 1988

Dissertation eingereicht am : 23.11.1988
Tag der mündlichen Prüfung : 21.12.1988
Referent : Prof. Dr.-Ing. W. Weber
Korreferent : Prof. Dr. J. Knop

Inhalt	Seite
1 Einleitung	5
1.1 Wünsche an ein Protokollspezifikationsverfahren	6
2 Protokolle in der Rechnerkommunikation	10
2.1 Protokolle	10
2.2 Schichtenstruktur	11
2.3 Kommunikation zwischen den Ebenen	15
3 Protokollbeschreibungsmethoden	17
3.1 Protokollentwurf im Gegensatz zum Software-entwurf	17
3.2 Prinzipieller Ablauf innerhalb eines Protokolls	19
3.3 Klassifizierung von Protokollbeschreibungsmethoden	22
3.3.1 Prinzipielle Auslegung	22
3.3.2 Charakteristische Merkmale	23
3.3.3 Funktionale Eigenschaften	24
3.3.4 Darstellungsform	26
3.3.5 Spezifikationsphilosophie	26
4 Bekannte Beschreibungsverfahren	29
4.1 Unstrukturierte Verfahren	29

4.2	Ereignisfolgenorientierte Verfahren	30
4.3	Zustandsorientierte Verfahren	32
4.4	Programmiersprachenähnliche Verfahren	37
4.5	Zusammenfassende Bemerkungen	39
5	Grobstruktur des Protokollentwurfsverfahrens	40
5.1	Prinzipielle Auslegung der Methode	41
5.2	Reaktionen auf Ereignisse	42
5.3	Prinzipielle Darstellungsmöglichkeiten	45
5.4	Repräsentation der Protokollspezifikation	51
6	Feinstruktur des Protokollentwurfsverfahrens	53
6.1	Definition der Zustände der Protokolleinheit	55
6.2	Definition von Konstanten und Variablen	56
6.3	Definition der Datenblöcke für die Protokolleinheit	60
6.4	Definition der auf das Protokoll wirkenden Ereignisse und der Schnittstellenaktionen	65
6.5	Zusammenhang zwischen den Datenblöcken und den unabhängigen Ereignissen und den Schnittstellenaktionen	69
6.5.1	Erkennen von unabhängigen Ereignissen	70
6.5.2	Auslösen von Schnittstellenaktionen	73
6.6	Vorverarbeitung von unabhängigen Ereignissen	78

6.7	Definition der Zustandsübergänge	85
6.8	Aktionen der Protokolleinheit	87
6.8.1	Zustandsunabhängige Aktionen	87
6.8.2	Zustandsabhängige Aktionen	92
6.9	Definition der Funktionen	96
6.10	Definition der Unterzustandsmaschinen	98
6.10.1	Weiterleiten von Ereignissen an Unterzustands- maschinen	102
6.10.2	Kommunikation zwischen Unterzustandsmaschine und übergeordneter Zustandsmaschine	103
7	Programmpaket zum Entwurf der Protokoll- spezifikation	108
7.1	Benutzeroberfläche des Programmpakets	108
7.2	Innere Struktur des Programmpakets	113
7.3	Syntaxüberprüfung der Protokollspezifikation	115
7.4	Simulation der Protokollspezifikation	117
8	Erfahrungen und Ausblick	122
9	Anhang	124
9.1	Struktur der zur Protokollspezifikation und -dokumentation verwendeten Listen	124
9.2	Literaturverzeichnis	154
9.3	Lebenslauf	165

Kurzfassung: Ein Protokollentwurfs- und -spezifikationsverfahren auf Basis einer erweiterten Zustandsdiagrammtechnik in Listenform mit Vorverarbeitung von Ereignissen

Ein unbefriedigend gelöstes Problem im Bereich der Rechnerkommunikation stellen die Protokollspezifikationen dar. So sind viele Protokolle, auch internationale Normen, in normaler Umgangssprache spezifiziert, wobei Teilaspekte dieser Protokolle mit unterschiedlichen, meist graphischen Darstellungen verdeutlicht werden.

Die Entwurfsmittel für Programme, die aus dem Bereich der Softwareentwicklung bekannt sind, sind für den Protokollentwurf nicht geeignet, da Protokolle nicht einem sequentiellen Ablauf folgen, sondern auf von außen einwirkende Ereignisse reagieren müssen, wobei gleichzeitig eine Gedächtnisfunktion für vergangene Ereignisse realisiert werden muß.

Dieser Sachverhalt kann mittels eines Zustandsdiagramms sehr gut dargestellt werden, jedoch kommt es bei komplexen Protokollen schnell zu einer Zustandsexplosion. Eine Abhilfe schafft hier die Einführung von Zustandsvariablen, die den Zustand näher beschreiben und zu einer Reduktion der Anzahl der Zustände führen.

Damit hängen die Reaktionen des Protokolls auf Ereignisse jedoch nicht mehr nur von den Zuständen ab, sondern auch von den Zustandsvariablen.

Eine Lösung stellt die Vorverarbeitung von Ereignissen dar. Dabei werden die auf das Protokoll wirkenden Ereignisse zunächst mit den Zustandsvariablen verknüpft, und daraus neue "abhängige" Ereignisse generiert. Diese abhängigen Ereignisse wirken auf das Zustandsdiagramm, wobei die Zustandsvariablen jetzt nicht mehr berücksichtigt werden.

Um den Protokollentwurf modular und hierarchisch gliedern zu können, wurden Unterzustandsmaschinen eingeführt, die in einem übergeordneten Zustand des Protokolls Teilabläufe selbsttätig abwickeln können.

Die Darstellung der Protokollspezifikation erfolgt in einer Listenform, um somit Rechnerunterstützung beim Entwurf des Protokolls, bei der Simulation der Protokollspezifikation und bei der Dokumentation zu erhalten. Diese Darstellungsform gewährleistet gleichzeitig eine exakte und eindeutige Protokollspezifikation, die sehr gut als Implementationsgrundlage geeignet ist.

1 Einleitung

Bei der Rechnerkommunikation haben lokale Netze und Weitverkehrsnetze inzwischen eine große Bedeutung erlangt. Im Bereich der lokalen Netze handelt es sich jedoch meist um private Netze, d.h. Rechnernetze, die nur auf Rechnern eines Herstellers betrieben werden. Hier hat jeder Hersteller seine eigenen Protokolle definiert, so daß ein Zusammenwirken mit anderen Rechnerarten überhaupt nicht oder nur über sogenannte Gateways möglich ist.

Das Entstehen oder auch die Verbreitung von herstellernabhängigen, also offenen Rechnernetzen, scheitert überwiegend daran, daß internationale Normen nicht für alle Bereiche verfügbar sind. In einigen Projekten, wie z. B. MAP (Manufacturing Automation Protocol) von General Motors oder TOP (Technical Office Protocol) von Boeing entstehen diese Normen zwar zur Zeit, sind aber gerade in den anwendernahen Schichten noch nicht genügend spezifiziert. Auch sehen diese Protokolle als Zielsysteme hauptsächlich Mini- und Großrechner mit multitaskfähigen Betriebssystemen vor. Im Bereich der Mikrorechner sind im wesentlichen jedoch nur private Netze vorhanden.

Im gesamten Bereich der Rechnerkommunikation sind also noch Protokolle zu entwerfen und zu spezifizieren, sei es als eigener Entwurf für eine offene Mikrorechnerkommunikation, oder als nationale und internationale Normen.

Der Entwurf eines Protokolls setzt natürlich, wie auch der Softwareentwurf /SNF79/, ein Entwurfsmittel voraus.

Aus welchen Gründen die bekannten Entwurfsverfahren für Programme auf den Protokollentwurf nicht anwendbar sind, wird in einem späteren Abschnitt dieser Arbeit aufgezeigt.

Zum besseren Verständnis der Wünsche und Anforderungen, die ein Anwender an eine Protokollspezifikationsmethode stellt, und der Gründe, die zum Entwurf eines neuen Protokoll-

spezifikationsverfahrens geführt haben, wird zunächst ein kurzer Einblick in diesen Bereich der Rechnerkommunikation gegeben.

1.1 Wünsche an ein Protokollspezifikationsverfahren

Die Bereiche, in denen ein Protokollbeschreibungsverfahren eingesetzt wird, lassen sich wie folgt angeben:

- Protokolldesign oder Protokollentwurf
- Protokollsimulation
- Protokollimplementation
- Protokolltest
- Protokolldokumentation oder Protokollspezifikation
(evtl. als Norm)

Die Wünsche desjenigen, der ein Protokoll entwirft, und somit in den meisten Fällen auch im Anschluß die Protokollspezifikation oder Dokumentation erstellt, an ein Protokollspezifikationsverfahren, lassen sich wie folgt charakterisieren.

Das Verfahren sollte in der Entwurfsphase eine Top-Down Vorgehensweise zulassen und somit hierarchische Strukturen unterstützen. Damit gleichsam verbunden ist eine Modularisierung des Entwurfs, der bei komplexen Protokollen erforderlich ist, um eine bessere Übersicht zu behalten. Das gesamte Protokollentwurfsverfahren sollte rechnergestützt arbeiten, damit der Anwender, möglichst selbsttätig aus der Entwurfsphase heraus, eine Protokollspezifikation und Dokumentation erhält. Änderungen an der Protokollspezifikation, bei festgestellten Fehlern im Protokoll, sollten schnell

durchführbar sein und ohne großen Aufwand zu einer neuen Dokumentation führen. Somit ergeben sich folgende

Wünsche beim Entwurf und der Protokollspezifikation:

- hierarchische Struktur
- Unterstützung von Modulen
- rechnergestütztes Verfahren
- möglichst selbsttätige Dokumentationserstellung
- einfache Änderungsmöglichkeiten.

Aufgabe der Protokollsimulation ist es eine Protokollspezifikation zu simulieren, um so festzustellen, ob diese möglichst fehlerfrei ist. Auf diese Weise sind in einem sehr frühen Stadium vor der Implementation mögliche Fehler im Protokoll zu erkennen. Um diesen Schritt durchführen zu können, ist eine sehr exakte und eindeutige Protokollspezifikation erforderlich. Diese Spezifikation sollte direkt von einer Rechenanlage interpretierbar sein, damit, ohne Umsetzung in eine andere Form, womit Übertragungsfehler ausgeschlossen werden, eine Simulation der Spezifikation durch einen Rechner ermöglicht wird.

Damit sind die

Wünsche bei der Protokollsimulation:

- exakte und eindeutige Protokollspezifikation
- Protokollspezifikation vom Rechner lesbar und interpretierbar.

Bei der Protokollimplementation geht es schließlich darum, das spezifizizierte Protokoll in irgendeiner Form zu realisieren. Dies kann in Form von Software, also als Programm, aber auch als Hardwarelösung in Form einer Schaltung aus dis-

kreten Bauteilen oder als integrierter Schaltkreis erfolgen. Eine wichtige Forderung ist hier abermals eine exakte und eindeutige Protokollspezifikation. Dies ist gerade bei internationalen Normen für offene Netze besonders wichtig, da diese von unterschiedlichsten Herstellern implementiert werden müssen. Bei Unklarheiten in der Protokollspezifikation kann dies zu unterschiedlichen Implementationen mit unterschiedlichen Funktionen der einzelnen Protokolle führen, womit eine Kommunikation zwischen diesen Systemen nicht zustande kommt.

Ein weiterer Wunsch an die Protokollspezifikation ist, daß sie unmittelbar als Implementationsgrundlage verwendet werden kann, ohne vorher nochmals auf ein anderes Entwurfsmittel überzugehen. Am wünschenswertesten wäre hier natürlich, wenn z. B. die Codegenerierung bei einer softwaremäßigen Implementation des Protokolls selbständig von einem Rechner auf Grund der Protokollspezifikation durchgeführt werden könnte oder dies zumindest in Teilen erfolgt. Somit ergeben sich die

Wünsche bei der Protokollimplementation:

- klare und eindeutige Protokollspezifikation
- Protokollspezifikation direkt als Implementationsgrundlage für Hard- und Softwarelösung verwendbar.

Der Test einer Protokolleinheit stellt eigentlich keine neuen Anforderungen an das Beschreibungsverfahren als diejenigen, die in den vorangegangenen Punkten schon aufgeführt sind.

Es gibt jedoch noch einige zusätzliche Wünsche an ein Protokollbeschreibungsverfahren, die allgemein in allen oben genannten Phasen gelten. Es sollte möglich sein, sich schnell

in die Protokollentwurfsmethode einzuarbeiten. Dies sollte an Stellen wo eine Rechnerunterstützung gegeben ist durch eine anwenderfreundliche menuegesteuerte Oberfläche unterstützt werden, die den Anwender durch die einzelnen Phasen des Protokollentwurfs führt und in der Lage ist, Hilfestellungen zu geben. Damit verbunden ist die Forderung nach einem klaren und durchstrukturierten Verfahren. Ein letzter Wunsch wäre, daß die Protokollspezifikation möglichst sprachenunabhängig ist, womit Verständnis oder Übersetzungsfehler in andere Sprachen ausgeschlossen werden. Zusammenfassend ergeben sich also folgende

allgemeine Wünsche:

- schnelle Einarbeitung in das Protokollentwurfs- und Spezifikationsverfahren
- sprachenunabhängige Protokollspezifikation
- klares und strukturiertes Verfahren.

2 Protokolle in der Rechnerkommunikation

Das folgende Kapitel beschreibt zunächst, um was es sich bei Protokollen handelt, und wie diese Protokolle in den gesamten Kommunikationsablauf eingebettet sind.

2.1 Protokolle

Der Datenaustausch in einem Rechnernetz muß bestimmten Regeln gehorchen, an die sich alle beteiligten Partner der Kommunikation halten. Jede Nachricht in einem Rechnernetz besteht daher aus zwei Komponenten (Bild 2.1-1), der Steuerinformation (äußere Nachricht) und der Nutzinformation

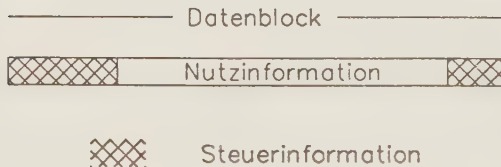


Bild 2.1-1 Datenblockaufbau in Rechnernetzen

(innere Nachricht). Die Steuerinformation kann Angaben über Attribute der Nachricht enthalten, wie z. B.

- Angabe des Übertragungsweges
- Kennzeichnung der Art der Information
- Daten zur Sicherung der Nachrichtenübertragung.

Die Vereinbarungen für den formalen Aufbau, die zeitliche Abfolge und die Interpretation der äußeren Nachricht werden als Protokoll bezeichnet.

Für die Abwicklung eines schnellen Datentransfers ist nicht nur eine hohe Datenrate auf dem Übertragungsmedium wichtig, sondern auch die Effizienz der daran beteiligten Protokolle von entscheidender Bedeutung. Der Planung und dem Entwurf dieser Kommunikationsprozeduren kommt daher eine große Bedeutung zu. Ein besonderes Augenmerk ist auf die Dokumentation und auf die eindeutige Beschreibung des Protokolls zu richten, da gerade bei internationalen Normen für offene Netze diese Beschreibung die Grundlage für eine Protokollimplementation bei den unterschiedlichsten Rechnerherstellern und Softwarehäusern ist. Ist die Protokollbeschreibung nicht eindeutig und verständlich, dann können beim Zusammenschluß unterschiedlicher Rechnersysteme zu einem Rechnernetz, verursacht durch unterschiedliche Protokollimplementationen, Protokollfehler auftreten, die bis hin zum totalen Zusammenbruch des gesamten Netzes führen.

2.2 Schichtenstruktur

Bereits die Implementation eines Protokolls für eine einfache Punkt zu Punkt Verbindung zwischen zwei Rechnern ist ein nicht triviales Problem. Die beiden Hosts müssen Daten über eine möglicherweise gestörte physikalische Verbindung austauschen. Sie sind trotz mehrfacher Quittungen, wie Schnupp zeigt /SCH82/, nie ganz sicher, in welchem Zustand sich der Kommunikationspartner befindet. Verlangt man von

dem Protokoll nicht nur einfache Basisdienste wie:

- gesicherter Datentransfer
- Auf- und Abbau der Verbindung
- Synchronisation der beiden Partner

sondern auch erweiterte Dienste wie:

- transparenter Datenaustausch
- effiziente Flußsteuerung
- Zeichensatz und Dateinamenkonvertierung

so ist leicht einzusehen, daß dieses Protokoll recht umfangreich wird. Geht man dann von einer Zweipunktverbindung zu einem vernetzten System über, so steigt die Komplexität der Protokolle nochmals. Es ist daher eine Modularisierung der kompletten Kommunikationsprozedur nötig.

Bei der Modularisierung der Kommunikationsprozedur hat sich eine Schichtenstruktur nach dem Geheimnisprinzip nach Parnas /PAR72/TH085/ durchgesetzt. Diese teilt die Gesamtaufgabe in mehrere Schichten oder Ebenen auf. Das Protokoll einer Ebene im Host A kommuniziert mit dem Protokoll der gleichen Ebene im Host B und wickelt so ein Schichtenprotokoll ab, das bestimmte Aufgaben der Kommunikation übernimmt. Dies geschieht jedoch nicht auf direktem Wege, sondern es bedient sich der Dienste der nächst tieferen Ebene, ohne zu wissen wie diese realisiert sind (Bild 2.2-1).

Nach welchen Gesichtspunkten diese Ebeneneinteilung erfolgen kann, zeigt Zimmermann /ZIM81/:

Eine Ebenengrenze sollte dort eingeführt werden, wo die Anzahl der Dienste der Ebene und der Informationsfluß zwischen den Ebenen minimal wird.

Eine neue Schicht sollte dann eingeführt werden, wenn

grundsätzlich andere Funktionen ausgeführt werden oder eine neue Abstraktionsebene erforderlich wird.

Jede Ebene muß eine wohldefinierte Funktion erfüllen, und ähnliche Funktionen sind in einer Ebene zu realisieren.

Die Anzahl der Schichten ist so groß zu wählen, daß unterschiedliche Funktionen in unterschiedlichen Ebenen realisiert werden, und so klein, daß die Gesamtübersicht nicht verloren geht.

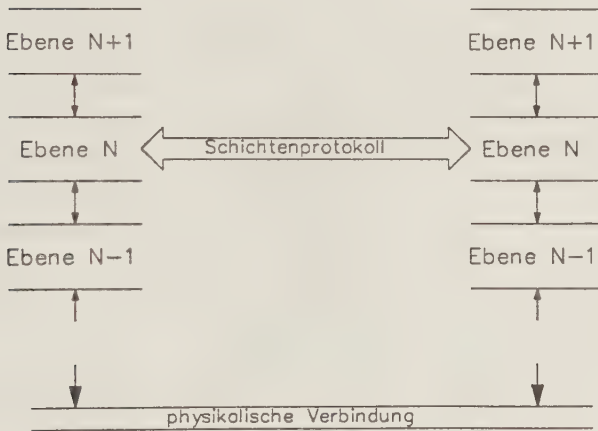


Bild 2.2-1 Schichtenprotokoll

Eine "offene" Kommunikation ist nur dann möglich, wenn Protokolle international genormt sind. Zu diesem Zweck hat die

ISO ein 7 - Schichten Architekturmodell vorgeschlagen (OSI Open Systems Interconnection) /TAN81/. Dieses Modell ist keine Norm für ein Protokoll, sondern es stellt einen Vorschlag dar, wie die Ebenenaufteilung erfolgen soll, und welche Funktionen die Protokolle der einzelnen Ebenen erfüllen sollen. Bild 2.2-2 zeigt das ISO/OSI Modell. Auf die Funktion der einzelnen Ebenen wird hier nicht näher eingegangen, da für eine Protokollspezifikationsmethode nur die Tatsache der Ebenenaufteilung, der Ablauf innerhalb des Protokolls und die Kommunikation zwischen benachbarten Ebenen von Bedeutung ist.



Bild 2.2-2 ISO/OSI 7 Schichtenmodell

2.3 Kommunikation zwischen den Ebenen

Da das Schichtenprotokoll zwischen zwei korrespondierenden Ebenen nur virtuell abgewickelt wird, muß das Protokoll einer Ebene sich der Dienste der nächst tieferen Schicht bedienen, um seine Nachrichten über das Netz zu versenden. Dazu bietet die tiefere Schicht einen oder mehrere Service Access Points (SAPs) an. Nur über diese Schnittstellen kann auf die Dienste der tieferen Schicht zugegriffen werden. Die Zugriffe der höheren (N+1) oder tieferen (N-1) Schicht erfolgen über einen oder mehrere SAPs der Ebene N. Die Ebene N selbst greift auf die Ebenen N+1 und N-1 nur über deren SAPs zu.

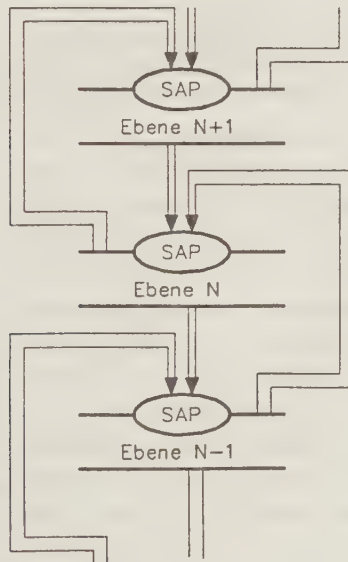


Bild 2.3-1 Kommunikation bei einem SAP pro Ebene

Bild 2.3-1 zeigt die prinzipielle Vorgehensweise bei einem SAP pro Protokolleinheit, Bild 2.3-2 wenn die Protokolleinheit mehrere SAPs zur Verfügung stellt.

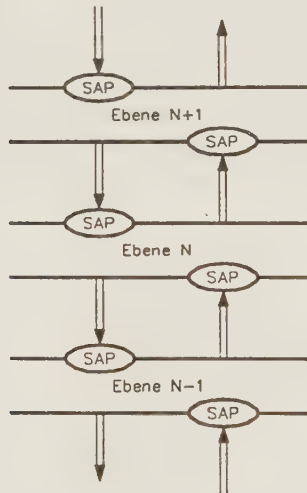


Bild 2.3-2 Kommunikation bei mehreren SAPs pro Ebene

Auf die Abläufe innerhalb einer Protokolleinheit, und wie das Protokoll auf die einwirkenden Ereignisse reagiert, wird in einem späteren Abschnitt dieser Arbeit noch ausführlich eingegangen.

3 Protokollbeschreibungsmethoden

Zunächst einmal muß man sich fragen, warum die bekannten Beschreibungsmethoden wie Programmablaufpläne, Nassi Shneidermann Diagramme usw., die für den Entwurf von Softwarepaketen zur Verfügung stehen, nicht auch für den Protokollentwurf eingesetzt werden. Dazu muß zuerst einmal der Unterschied zwischen einem "normalen" Programm und einem Protokoll, das natürlich auch als Programm realisiert werden kann, dargestellt werden. Zwecks einfacherer Schreibweise werden im folgenden "normale" Programme einfach mit Programm bezeichnet, wenn der Begriff im Gegensatz zu Protokollen benutzt wird.

3.1 Protokollentwurf im Gegensatz zum Programmmentwurf

Betrachtet man einmal die Umgebung unter der ein Programm auf einem Rechner abläuft, so handelt es sich hier um einen Prozessor mit einem Betriebssystem, das dem Anwenderprogramm eine Reihe von Diensten zur Verfügung stellt. Das Programm läuft auf dem Rechner sequentiell ab. Werden Dienste vom Betriebssystem gefordert, so wird mit der Programmausführung gewartet, bis diese zur Verfügung gestellt sind. Die Synchronisationsmechanismen, die hier benötigt werden, sind aus diesem Grunde einfach und überschaubar. Selbst beim Übergang zu Mehrbenutzersystemen, wo mehrere Programme quasi gleichzeitig abgearbeitet werden, bleibt das Ganze überschaubar, da die Programme stückchenweise nacheinander abgearbeitet werden, und beim Umschalten auf ein anderes Programm der gesamte Zustand des Prozessors für das Programm gespeichert wird, so daß beim Weiterführen des suspendierten Programms der Originalzustand wieder hergestellt werden kann. Der Zustand, in dem sich ein Programm befindet, kann jederzeit

festgestellt werden, und Programmfehler können vom Betriebssystem abgefangen werden, so daß es durch das fehlerhafte Arbeiten eines Programms nicht zum Zusammenbruch des gesamten Rechners kommt.

Im Gegensatz dazu laufen Protokolle immer auf mehreren Rechnern gleichzeitig ab. Hierbei handelt es sich um eine echte Parallelität, da die einzelnen Rechner voneinander unabhängig arbeiten. Die Datenblöcke, die ausgetauscht werden, können bei der Übertragung verlorengehen oder verfälscht werden, und selbst bei mehrfachem Austausch von Quittungen bleibt eine Restunsicherheit, in welchem Zustand sich der Kommunikationspartner befindet /WEB84/. Daraus ergibt sich, daß der Zustand des gesamten Netzes nicht bekannt ist, und die einzelnen Protokolleinheiten auftretende Fehler im Netz abfangen müssen, da sonst die gesamte Kommunikation in dem Netzwerk zusammenbrechen kann. Ein weiteres Problem stellt die Synchronisation der einzelnen Protokolleinheiten dar, da sie auf getrennten Rechnern eigenständig ablaufen, und Steuerinformation zum Synchronisieren nur über eine möglicherweise gestörte Datenleitung austauschen können.

Die vorgenannten Überlegungen haben zur Folge, daß die Entwurfsverfahren für Programme auf einen sequentiellen Ablauf ausgerichtet sind. Beim Aufruf von Unterprogrammen oder Modulen werden diese ebenfalls sequentiell abgearbeitet und hinter der aufrufenden Stelle mit der Programmausführung fortgefahren. Der Programmablauf ist bekannt, und notwendige Synchronisation, z. B. bei Ein- Ausgabe Operationen, wird durch das Betriebssystem geleistet.

Die Entwurfsverfahren für Protokolle müssen auf das asynchrone Eintreten von Ereignissen ausgerichtet sein. Es gibt hier keinen sequentiellen Ablauf mehr, sondern der Protokollablauf wird alleine durch das Eintreten bestimmter Ereignisse von außen vorgegeben, und die Reaktionen des

Protokolls hängen von diesen und vergangenen Ereignissen, die auf das Protokoll gewirkt haben, ab. Ein Entwurfsmittel für Protokolle muß daher auf diese Gegebenheiten besondere Rücksicht nehmen.

3.2 Prinzipieller Ablauf innerhalb eines Protokolls

Bevor nun näher auf Protokollspezifikations- und Dokumentationsverfahren eingegangen wird, zunächst noch ein paar prinzipielle Vorbemerkungen zum Ablauf in einer Protokolleinheit. Protokolle werden nur dann zu Aktionen veranlaßt, wenn sie durch asynchron eintretende Ereignisse angestossen werden. Die Ereignisse, die auf eine Protokolleinheit der Ebene N einwirken, können wie folgt angegeben werden:

- A Anforderungen der Ebene N+1 um Dienste der Ebene N in Anspruch zu nehmen.
- M Meldungen der Ebene N-1 an die Ebene N.
- B Bestätigungen der Ebene N+1 auf Meldungen von der Ebene N.
- R Antworten (Responses) der Ebene N-1 auf Anforderungen durch die Ebene N.

Zusätzlich zu diesen Ereignissen, die alle von außen auf die Protokolleinheit einwirken, gibt es noch ein mögliches weiteres Ereignis:

- I Inneres Ereignis der Ebene N

Diese inneren Ereignisse einer Protokolleinheit sind abgelaufene Zeitüberwachungen, die das Protokoll vor Hängenbleiben aufgrund von ausgebliebenen oder verfälschten erwarteten äußeren Ereignissen schützen soll. Bild 3.2-1 zeigt die auf die Protokolleinheit wirkenden Ereignisse.

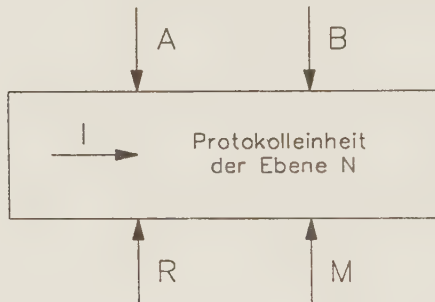


Bild 3.2-1 Ereignisse für die Protokolleinheit

Äußere Ereignisse wirken nur an den Service Access Points der Protokolleinheit. Das Ereignis selbst besteht in dem Eintreffen eines Datenblockes. Damit wird die Protokolleinheit an einer zentralen Stelle gestartet, und überprüft um welches Ereignis, das im Datenblock verschlüsselt ist, es sich handelt. Dieses Ereignis wird entsprechend seiner Bedeutung verarbeitet und veranlaßt die Protokolleinheit schließlich dazu, eigene Aktionen auszulösen. Die möglichen Aktionen einer Protokolleinheit lassen sich wie folgt angeben:

- A Anforderung an die Ebene N-1 um deren Dienste in
 Anspruch zu nehmen.
- M Meldungen an die Ebene N+1
- R Antworten (Responses) an die Ebene N+1 für erhal-
 tene Anforderungen
- B Bestätigungen an die Ebene N-1 für empfangene
 Meldungen

Bild 3.2-2 zeigt die Aktionen der Protokolleinheit der Ebene N. Alle Aktionen können nur in Form von Datenblöcken an die SAPs der Ebenen N-1 und N+1 gesendet werden.

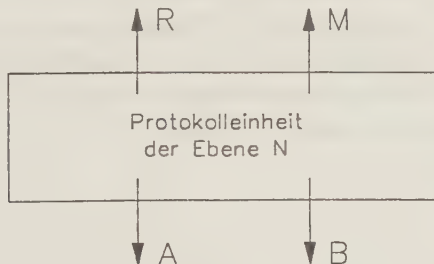


Bild 3.2-2 Aktionen der Protokolleinheit

3.3 Klassifizierung von Protokollbeschreibungsmethoden

Um einen Überblick über bereits bekannte Protokollspezifikationsverfahren zu erhalten, wird zunächst eine Klassifizierung von Beschreibungsmethoden vorgenommen, wobei nach charakteristischen Eigenschaften der Methoden unterschieden werden soll.

Eine erste Einteilung kann vorgenommen werden, indem zwischen

- unstrukturierten Beschreibungsverfahren und
- strukturierten Beschreibungsverfahren

unterschieden wird.

Zu den unstrukturierten Verfahren zählt die natürliche Sprache. Hier wird versucht die Protokolleinheit und die Abläufe innerhalb der Protokolleinheit mittels normaler Umgangssprache zu beschreiben. Dies führt bei komplexen Protokollen zu teilweise schwer überschaubaren Spezifikationen, die dann auch nicht immer eindeutig die Funktion des Protokolls wiedergeben. Die Protokollspezifikation ist schlecht als Implementationsgrundlage zu verwenden, da hier formale Methoden benötigt werden.

3.3.1 Prinzipielle Auslegung

Die strukturierten Beschreibungsverfahren lassen sich weiter aufteilen, und zwar zunächst einmal nach der prinzipiellen Auslegung der Methode. Es wird zwischen

- zustandsorientierten Verfahren und
- ereignisfolgenorientierten Verfahren

unterschieden.

Ereignisfolgenorientierte Verfahren können dadurch charakterisiert werden, daß die Reaktionen einer Protokolleinheit auf ein bestimmtes von außen einwirkendes Ereignis durch die Gesamtheit aller bisher eingetretenen Ereignisse bestimmt sind. Die Beschreibung der Protokolleinheit geschieht hier durch die Definition aller zulässigen Ereignisfolgen. Spezifikationsmethoden dieser Art sind eigentlich nur dann sinnvoll anwendbar, wenn zur Beschreibung des Protokolls wenige kurze Ereignisfolgen ausreichen.

Bei den zustandsorientierten Verfahren werden Zustände definiert, in denen sich eine Protokolleinheit befinden kann. Die Reaktion auf ein von außen einwirkendes Ereignis hängt hier nur von dem gerade aktuellen Zustand und dem eingetretenen Ereignis selbst ab. Dabei ist es gleichgültig auf welche Art und Weise die Protokolleinheit in diesen Zustand gelangt ist. Nachdem das Protokoll entsprechend dem Ereignis und dem aktuellen Zustand bestimmte Aktionen ausgeführt hat, wird ein Zustandswechsel vorgenommen. Die Beschreibung der Protokolleinheit erfolgt also durch Definition von Zuständen und Zustandsübergängen, die von Ereignissen, die auf das Protokoll einwirken, bewirkt werden. Für jeden Zustand muß die Reaktion auf jedes Ereignis definiert sein.

3.3.2 Charakteristische Merkmale

Für die strukturierten Verfahren können nun noch weitere charakteristische Merkmale angegeben werden /PIA86/. Zum einen unterscheidet man, ob das Verfahren in der Lage ist Protokolle zu beschreiben, die aus nur einer Komponente oder einem Modul bestehen, oder ob die Beschreibungsmethode es

zuläßt Protokolle zu definieren, die sich aus mehreren Komponenten zusammensetzen.

Bei den Verfahren, die Protokolle, die sich aus mehreren Komponenten zusammensetzen, beschreiben, unterscheidet man zusätzlich, ob die einzelnen Komponenten nur sequentiell nacheinander, oder aber parallel ablaufen können. Der Informationsaustausch zwischen den einzelnen Komponenten kann entweder separat definiert werden, oder aber integraler Bestandteil der Spezifikation sein.

3.3.3 Funktionale Eigenschaften

Die einzelnen Komponenten können noch näher beschrieben werden, wobei funktionale Eigenschaften der Methode angegeben werden. Die wichtigsten sind im folgenden kurz aufzeigt:

- kombinatorische Funktion

Hierbei handelt es sich um ein Verfahren, wobei die Reaktion und die Ausgaben einer Protokolleinheit nur von den gerade aktuellen Ereignissen oder Eingängen, die auf das Protokoll wirken, abhängen. Dieser Zusammenhang kann als

$$\text{Reaktion} = f(\text{aktuelles Ereignis})$$

beschrieben werden. Dabei ist es nicht möglich eine Gedächtnisfunktion zu realisieren, da vergangene Ereignisse oder Zustände des Protokolls vollkommen unberücksichtigt bleiben.

- diskrete Zustandsmaschine

Bei der Beschreibung eines Protokolls als Zustandsmaschine besitzt das Protokoll ein Gedächtnis für vergangene Aktionen und Ereignisse, die durch einen Zustand beschrieben sind. Jede Änderung im Status des Protokolls muß durch einen neuen Zustand der Zustandsmaschine beschrieben werden. Die Reaktion des Protokolls hängt jetzt vom aktuellen Zustand und von dem eingetretenen Ereignis ab. Dies kann angegeben werden als

$\text{Reaktion} = f(\text{aktuelles Ereignis}, \text{aktueller Zustand})$.

Ist die Anzahl der Zustände endlich, so spricht man von einer endlichen Zustandsmaschine.

- Warteschlange

Eine Warteschlange ist in der Lage eingehende Daten zwischenspeichern und auf ein Ereignis hin, ein gespeichertes Element wieder auszugeben. Die Reihenfolge ist dabei, daß zuerst gespeicherte Daten auch zuerst ausgegeben werden (FIFO).

- programmierbarer Timer

Ein Timer ist eine ablaufende Uhr, die, wenn sie abgelaufen ist, ein Ereignis auslöst. Das Ablaufen eines Timers erfolgt vollkommen selbständig, sobald er einmal gestartet wurde.

- Delay

Hierbei handelt es sich um eine einfache Zeitverzögerung von Daten oder sonstigen Eingängen. Dieser Vorgang kann angegeben werden als

$\text{Ausgabe} = \text{Eingabe} + \text{Zeitintervall}$.

3.3.4 Darstellungsform

Ein weiteres Unterscheidungsmerkmal bei einem Protokollspezifikationsverfahren ist die Art und Weise der Darstellung der Spezifikation. Hier sind zunächst drei prinzipielle möglich Arten zu unterscheiden:

- graphische Notation
- zeichenorientierte Darstellung
- ein Gemisch aus zeichenorientierter und graphischer Darstellungsweise

Die graphische Darstellungsweise hat den Vorteil, daß man sich sehr schnell einen Überblick über die Funktionsweise der Protokolleinheit verschaffen kann, solange die Darstellung übersichtlich bleibt. Der Vorteil einer zeichenorientierten Darstellung liegt in der Möglichkeit, daß die Spezifikation maschinenlesbar ist, d.h. von einem Rechner verarbeitet werden kann.

3.3.5 Spezifikationsphilosophie

Als letztes Beschreibungsmerkmal soll nun die Spezifikationsphilosophie der Beschreibungsmethode betrachtet werden. Es wird zwischen zwei prinzipiellen Möglichkeiten,

- der Metaimplementation und
- dem Monitor

unterschieden. Betrachtet man zwei Systeme, die über ein Netz kommunizieren sollen (vgl. Bild 3.3.5-1), so beschreibt die Metaimplementation die Protokollsysteme direkt, als ob die Spezifikation anstelle einer tatsächlichen Implementa-

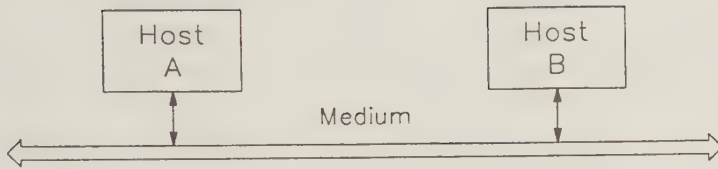


Bild 3.3.5-1 Kommunikation zweier Systeme über ein Netz

tion des Protokolls stehen würde (Bild 3.3.5-2).

Die einwirkenden Ereignisse und die Aktionen sind die, die bei einer späteren Implementation auch auf die Protokoll-einheit einwirken.

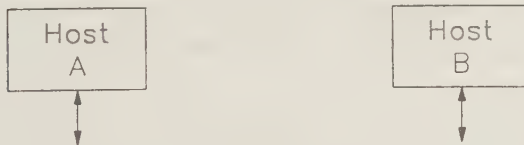


Bild 3.3.5-2 Metaimplementation

Anders ist es bei der Monitor Spezifikationsphilosophie. Hier wird der Datenfluß zwischen den beiden kommunizierenden Systemen beschrieben (Bild 3.3.5-3).

Die Monitor Philosophie liefert eine indirekte Protokoll-

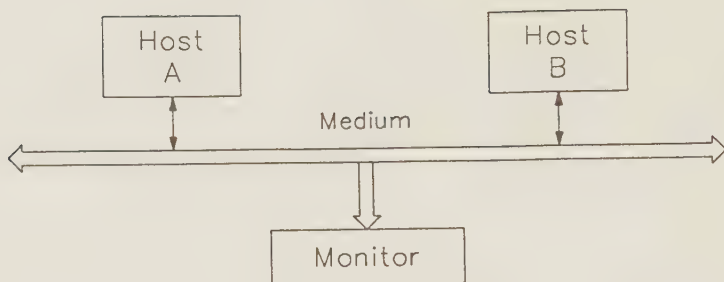


Bild 3.3.5-3 Monitor-Methode

beschreibung, wobei Aktions- und Ereignisfolgen an den Schnittstellen der Protokolle angegeben werden, so daß für die Kommunikationspartner legale Aktions- und Ereignisfolgen bekannt sind. Aus diesen bekannten Folgen können dann Rückschlüsse auf den inneren Aufbau der Protokolleinheit gezogen werden.

Beide Spezifikationsphilosophien, die Metaimplementation und die Monitor Methode, lassen sich sowohl mit zustandsorientierten Verfahren als auch mit ereignisfolgenorientierten Verfahren realisieren.

4. Bekannte Beschreibungsverfahren

Im folgenden Abschnitt werden einige bereits bekannte und veröffentlichte Protokollbeschreibungsverfahren vorgestellt. Dies kann jeweils nur exemplarisch für bestimmte Klassen, in die sich die Beschreibungsmethoden einteilen lassen, geschehen, sollte jedoch einen guten Überblick über die bekannten Entwurfswerkzeuge für Protokolle geben. Es wird versucht die Verfahren in die folgenden vier Gruppen einzuteilen:

- Unstrukturierte Verfahren
- Ereignisfolgenorientierte Verfahren
- Zustandsorientierte Verfahren
- Programmiersprachenähnliche Verfahren

Diese Einteilung gelingt nicht immer, da einige Beschreibungsmethoden mehrere Möglichkeiten unterstützen.

4.1 Unstrukturierte Verfahren

Die am häufigsten benutzte Methode um irgendwelche Vorgänge und Zusammenhänge darzustellen, ist die natürliche Sprache. Internationale Normen, wie zum Beispiel die Protokolle der X-Serie X.1 - X.29 des CCITT /YEL81a/, sind in englischer Sprache abgefaßt. Bei technischen Problemen, also auch bei Protokollen, ist es jedoch schwierig, komplizierte Vorgänge gleichzeitig klar und einfach wie auch eindeutig zu beschreiben.

Versucht man komplexe Protokolle in Umgangssprache zu formulieren, so führt dies zu umfangreichen Texten, die durch ihre Größe unübersichtlich sind, und somit zu Unverständlichkeiten in der Protokollbeschreibung führen. Ein weiteres Problem stellen Übersetzungen in andere Sprachen dar, wo-

durch sich Fehler in die Protokollspezifikation einschleichen, was dann zu unterschiedlichen Protokollimplementationen führen kann. Eine in Umgangssprache abgefaßte Protokollspezifikation ist, außer im Sinne der Textverarbeitung, nicht von einer Rechenanlage verarbeitbar oder interpretierbar. Als Implementationsgrundlage ist die natürliche Sprache nicht zu verwenden, da hier formale Verfahren benötigt werden.

4.2 Ereignisfolgenorientierte Verfahren

Ereignisfolgenorientierte Verfahren beschreiben eine Protokolleinheit, indem sie für das Protokoll legale Folgen von Ereignissen definieren. Beschränkt man sich hier auf rein ereignisfolgenorientierte Verfahren und verzichtet ganz auf die Einführung von Zuständen, so ergeben sich bei etwas größeren Protokollen sehr viele und sehr umfangreiche Folgen. Als Beispiel für dieses Verfahren seien hier die im deutschen Sprachraum häufig verwendeten Linien oder Raum-Zeit Diagramme angeführt, die unter anderem bei der Beschreibung des Datex-P Dienstes /DAT83/, bei den EHKPs (einheitliche höhere Kommunikationsprotokolle) /EH483/EH583/ oder bei der Beschreibung des ISDN-Dienstes /BOC86/ verwendet werden.

Raum-Zeit Diagramme bieten die Möglichkeit parallele Ereignisse und Abläufe in Sender und Empfänger darzustellen. Sie geben die zeitliche Abfolge der Ereignisse wieder, sind jedoch nicht in der Lage auf unterschiedliche Ereignisse zu reagieren. Dazu muß eine neue Ereignisfolge angegeben werden. Bestünde in einer Folge z. B. an zwei Stellen die Möglichkeit, daß zwei unterschiedliche Ereignisse auf die Protokolleinheit treffen, so müssen dazu schon vier Ereignis-

folgen spezifiziert werden.

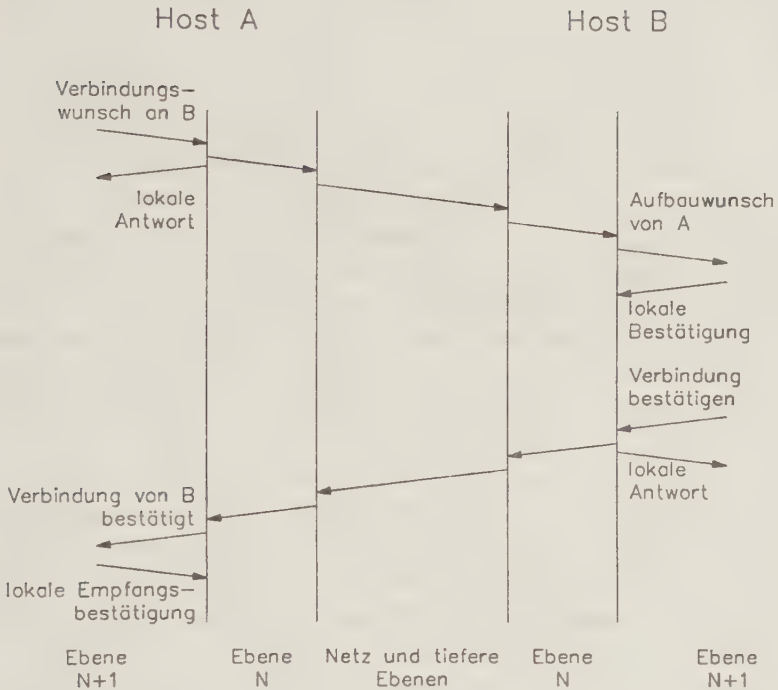


Bild 4.2-1 Linien oder Raum-Zeit Diagramm

Bild 4.2-1 zeigt ein solches Linien oder Raum-Zeit Diagramm. Die Darstellungsweise ist graphisch, und nicht von Rechnern verarbeitbar. Der Haupteinsatz für diese Beschreibungs-methode ist in kleinen Protokollen oder in kleinen Teilaspek-ten eines Protokolls, z. B. beim Verbindungsaufbau zu fin-

den, wo der gegenseitige Austausch und die zeitliche Abfolge von Informationen zwischen zwei Protokolleinheiten verdeutlicht werden soll.

4.3 Zustandsorientierte Verfahren

Bei sämtlichen Protokollbeschreibungsmethoden, die auf zustandsorientierten Verfahren basieren, wird der aktuelle Status einer Protokolleinheit in einem Zustand festgehalten. Damit wird, wie schon früher in dieser Arbeit erwähnt, erreicht, daß die Aktionen des Protokolls nur noch von dem jeweiligen Ereignis und dem gerade aktuellen Zustand abhängen.

Die einfachste Form für ein solches Verfahren ist eine Zustandsmaschine. Bei einer graphischen Darstellungsweise können die Zustände als Kreise symbolisiert werden, und die Zustandsübergänge als Kreisbögen zwischen den einzelnen Zuständen. An jeden Kreisbogen werden das Ereignis, das den Zustandsübergang bewirkt, und die Aktionen des Protokolls bei diesem Übergang geschrieben. In Bild 4.3-1 ist ein Beispiel für eine endliche Zustandsmaschine angegeben. Obwohl es sich in diesem Beispiel nur um einen sehr einfachen Teilaspekt eines Protokolls handelt, lassen sich doch schon die Beschränkungen dieses Verfahrens erkennen. Zum einen sind umfangreichere Protokollaktionen nicht mehr an den Kreisbögen anzugeben. Zum anderen liegt ein großer Nachteil dieses Verfahrens darin, daß es sehr schnell zu einer Explosion der Zustände kommen kann. Soll ein Protokoll beispielsweise 10 Sendeveruche bei einer fehlerhaften Blockübertragung zulassen, so resultiert dies in 10 unterschiedlichen Zuständen, die sich jedoch nur durch den aktuellen Sendeveruch unterscheiden. Deshalb finden reine Zustandsmaschinen nur bei

sehr kleinen Protokollen, wie z. B. die X.20 oder X.21 Empfehlungen des CCITT, Verwendung /TIE82/.

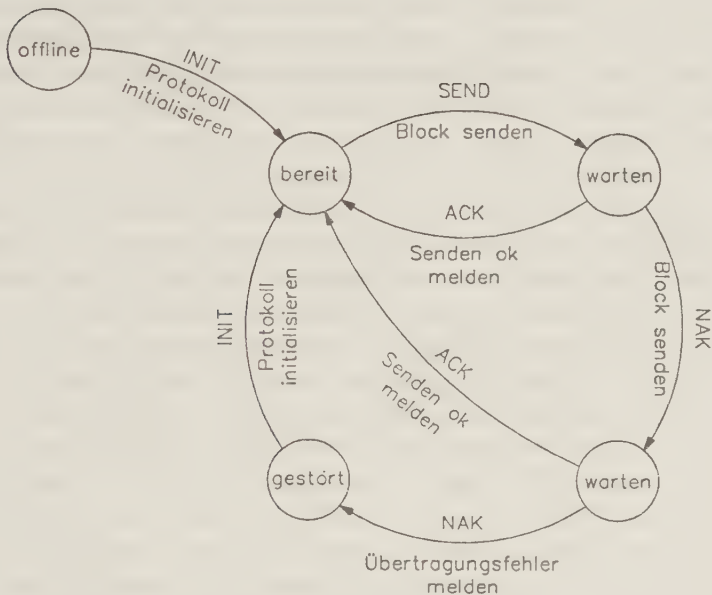


Bild 4.3-1 endliche Zustandsmaschine

Die Einführung von Zustandsvariablen führt zu einer erweiterten Zustandsdiagrammtechnik. Der Status des Protokolls ist nun nicht mehr alleine von dem aktuellen Zustand, sondern auch von den ihn näher beschreibenden Zustandsvariablen, abhängig. Der Status des Protokolls kann jetzt durch einen Zustandsvektor, gebildet aus Zustand und Zustands-

variablen, beschrieben werden.

Diese erweiterten Zustandsdiagrammtechniken sind in der Literatur in vielfältiger Form anzutreffen. So beschreiben Simon und Kaufman /SIM82/ ein Verfahren, das in einem Zustand einer Zustandsmaschine eine untergeordnete Zustandsmaschine definiert. Dies führt zu einer hierarchischen Strukturierung des Spezifikationsverfahrens. Die Darstellung der Spezifikation erfolgt in einer programmiersprachenähnlichen Form.

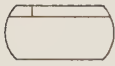
Weitere Verfahren sind in internationalen Normen zu finden. In der CSMA/CD Spezifikation /802.385/ wird im Zustandsdiagramm jeweils nur das Ereignis, das einen Zustandsübergang bewirkt, angegeben. Eine nähere Beschreibung der Aktionen, der Ereignisse und der einzelnen Zustände erfolgt in umgangssprachlicher Form. Eine andere Form, wo versucht wird den Ablauf der Zustandsübergänge näher zu beschreiben, indem die Aktionen der Protokolleinheit und die eintretenden Ereignisse im Diagramm angegeben werden, findet man in der Token-Ring Spezifikation /802.585/. Auch hier werden die einzelnen Zustände, Ereignisse und Aktionen in umgangssprachlicher Form näher beschrieben, da die graphische Darstellung zur eindeutigen Spezifikation nicht ausreicht.

Als Implementationsgrundlage sind die graphischen Repräsentationen der Zustandsdiagramme eine Hilfe, da man schnell einen Überblick über die prinzipielle Funktion des Protokolls erhält. Da aber bei komplexeren Protokollen nicht mehr alle Aktionen der Protokolleinheit, und in vielen Fällen auch nicht mehr die Ereignisse unter Berücksichtigung der Zustandsvariablen in das Zustandsdiagramm eingetragen werden können, reicht diese Form der Protokollspezifikation alleine nicht aus.

Eine andere, ebenfalls zustandsorientiert arbeitende Methode, sind Petri-Netze. Dieses graphische Verfahren kann zur Spezifikation von Protokollen herangezogen werden. Dabei können die Plätze in Petri-Netzen mit den Zuständen eines Zustandsdiagramms verglichen werden. Der Vorteil von Petri-Netzen gegenüber Zustandsdiagrammen ist, daß sich mit Hilfe der Petri-Netze beide an einer Kommunikation beteiligten Partner gleichzeitig darstellen lassen. Da bei dieser Methode mehrere Prozesse gleichzeitig aktiv sein können, kann hier durch ein sogenanntes Markenspiel /ZUS80/ die Konfliktfreiheit der Protokollspezifikation überprüft werden. Die reinen Petri-Netze sind für die Protokollspezifikation, ähnlich wie die Zustandsdiagramme, erweitert worden. Ein Beispiel für "timed Petri-Nets" beschreibt Walter /WAL83/, wobei die Zeitabhängigkeiten des Protokolls mit in das Modell eingebracht werden. Eine andere Darstellungsweise zeigt Billington /BIL83/ mit bezeichneten numerischen Petri-Netzen (labelled numerical Petri nets).

Obwohl die vorgenannten graphischen Methoden einen schnellen Einblick in die Funktionsweise eines Protokolls vermitteln, so werden sie bei umfangreicheren Protokollen doch sehr schnell unübersichtlich. So stellt auch Sunshine in einem Aufsatz über Spezifikationsmethoden fest, daß die Einschränkungen von Zustandsdiagrammen und Petri-Netzen bedeuten, daß diese Entwurfsmittel nur bei einfachen Protokollen oder auf Teilaspekte von etwas komplexeren Protokollen anwendbar sind /SUN81/.

Ein weiteres graphisches Verfahren ist SDL (Functional Specification and Description Language), vorgeschlagen vom CCITT in den Empfehlungen Z.101 - Z.104 /YEL81b/. Dieses Verfahren unterscheidet Zustände, Ereignisse und Aktionen



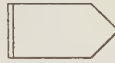
Zustands-
Symbol



Aufbewahrungs-
Symbol



Empfang
intern



Senden
intern



Empfang
extern



Senden
extern



Ausführungs-
Symbol



Entscheidungs-
Symbol



Fluß-Linie



Zusammenführung

Beispiel



Verbindungspunkt



Aufspaltung

Beispiel

Bild 4.3-2 SDL Grafiksymbole

durch unterschiedliche Grafiksymbole (vgl. Bild 4.3-2). Diese Methode läßt es zu, ein Protokoll sowohl zustandsorientiert als auch ereignisfolgenorientiert zu beschreiben. Die Diagramme werden jedoch, wie die Beispiele in den CCITT Empfehlungen zeigen, schon bei kleineren Protokollen recht umfangreich und unübersichtlich. Das Verfahren unterstützt keine hierarchischen Strukturen und beinhaltet Elemente aus der Flußdiagrammtechnik, die beliebige Entwurfsstrukturen zulassen, und ein strukturiertes Vorgehen beim Protokollentwurf nicht unterstützen.

Als letztes zustandsorientiertes Verfahren seien noch die von Schnupp beschriebenen Interaktionsdiagramme erwähnt /SCN82/. Diese ebenfalls graphische Methode benutzt ebenfalls Elemente aus der Flußdiagrammtechnik und stellt die Zustände der Protokolleinheit durch unterschiedliche graphische Symbole dar, wodurch die Bedeutung eines Zustands näher erläutert werden soll. Das Verfahren arbeitet hierarchisch, wobei die einzelnen Grafikelemente durch umfangreiche Texte ergänzt werden.

4.4 Programmiersprachenähnliche Verfahren

Zum Abschluß der Vorstellung von Protokollbeschreibungsmethoden sollen programmiersprachenähnliche Verfahren vorgestellt werden. Diese Verfahren können in zwei Klassen eingeteilt werden, und zwar in

- Programmiersprachen und
- spezielle Protokollprogrammiersprachen.

Diese Methoden geben eine sehr exakte Beschreibung des Protokolls, und man findet sie oft in Normenspezifikationen. So wird das CSMA/CD Protokoll /802.385/ außer in einer Zustandsmaschinendarstellung zusätzlich in Pascal spezifiziert und das Token-Passing-Bus Protokoll in einer Pascal ähnlichen Form /802.485/. Eine Möglichkeit Protokolle in Modula 1 zu spezifizieren beschreibt Schicker /SCI83/.

Programmiersprachenähnliche Verfahren können sowohl zustandsorientiert wie auch ereignisfolgenorientiert arbeiten, da ja jedes Protokoll in Form eines Programms, in einer Programmiersprache geschrieben, realisiert und implementiert werden kann. Dies läßt diese Verfahren für den Protokollentwurf jedoch wenig sinnvoll erscheinen, da ein Programm wiederum selbst ein Entwurfsmittel voraussetzt.

Um auf die besonderen Erfordernisse bei Protokollen besser eingehen zu können, wurden auch mehrere spezielle Protokollprogrammiersprachen entwickelt. Eine Sprache, die auf Pascal basiert, ist das Ergebnis einer Arbeitsgruppe der ISO (ISO/SCI6/WG1/FDT) und nennt sich PDIL (Protocol Description and Implementation Language). Die Sprache basiert auf der kombinierten Benutzung von Zustandsmaschinen und Petri-Netzen /ANS82/. Die Syntax von PDIL ist sehr Pascal ähnlich, wobei Elemente hinzugefügt wurden, die auf die speziellen Protokollanforderungen, wie das asynchrone Eintreten von Ereignissen, besondere Rücksicht nehmen /VAL85/. Die Sprache unterstützt sowohl die Implementation wie auch die Simulation des spezifizierten Protokolls.

Eine andere Sprache, die speziell auf Zustandsmaschinen abgestimmt ist, ist SL1 /EXE82/. Die Syntax der Sprache entspricht der CCITT Hochsprache CHILL /REK82/. Es ist eine sehr einfache Sprache, und sie trennt nicht zwischen Protokollentwurf und Protokollimplementation.

In der Literatur finden sich noch mehrere programmiersprachenähnliche Verfahren, die hier jedoch nicht mehr aufgezeigt werden sollen.

4.5 Zusammenfassende Bemerkungen

Betrachtet man die hier vorgestellten Entwurfsverfahren unter Berücksichtigung der im ersten Abschnitt dieser Arbeit aufgestellten Wünsche und Forderungen, so wird für den Protokollentwurf ein formales und strukturiertes Verfahren benötigt. Die Spezifikation sollte von einem Rechner verarbeitet werden können, um die Simulation und die Implementation zu unterstützen. Die vorgestellten graphischen Verfahren werden schnell unübersichtlich, und kommen, da nicht alle Informationen in die graphische Darstellung eingebracht werden können, ohne nähere Beschreibung, meist in umgangssprachlicher Form, nicht aus. Auf der anderen Seite repräsentieren programmiersprachenähnliche Verfahren eigentlich schon eine Implementation des Protokolls, und sind kein Entwurfsmittel, da Programme ja selbst wiederum ein Entwurfsmittel voraussetzen /GEW85/. Diese Überlegungen haben dazu geführt, eine formale, strukturierte und hierarchische Entwurfsmethode für Protokolle zu spezifizieren.

5 Grobstruktur des Protokollentwurfsverfahrens

Die Anforderungen an das Protokollspezifikationsverfahren ergeben sich aus den in Kapitel 1.1 aufgezeigten Wünschen in den unterschiedlichen Einsatzbereichen der Spezifikationsmethode. Diese Forderungen werden im folgenden nochmals kurz aufgezeigt:

- Das Entwurfsverfahren und die daraus entstehende Protokollspezifikation sollen durchgehend, in allen Phasen der Protokollentwicklung, vom Entwurf des Protokolls, über die Spezifikation und Dokumentation, die Simulation der Spezifikation, bis zur Implementation in Form von Hard- oder Software, ohne Zwischenschritte oder die Benutzung anderer Verfahren, anwendbar sein.
- Die Methode soll eine hierarchische Aufteilung der Gesamtaufgabe in Module zulassen, wobei nach einem Top - Down - Verfahren vorgegangen werden soll.
- Die Protokollspezifikation soll von einem Rechner lesbar und interpretierbar sein, um eine Simulation der Spezifikation durchführen zu können, und um Rechnerunterstützung beim Entwurf, der Dokumentation und der Implementation zur Verfügung stellen zu können.
- Die Spezifikation muß exakt und eindeutig und möglichst sprachenunabhängig sein.
- Für eine schnelle Einarbeitung in das Entwurfsverfahren wird eine klare und strukturierte Methode gefordert.

5.1 Prinzipielle Auslegung der Methode

Beim Entwurf eines Protokollspezifikationsverfahrens muß zunächst einmal die prinzipielle Auslegung des Verfahrens festgelegt werden. Unstrukturierte Verfahren, wie die natürliche Sprache, scheiden von vornherein aus, da sie zu ungenau und unübersichtlich sind und sich weder von einem Rechner verarbeiten lassen, noch als Implementationsgrundlage zu gebrauchen sind.

Bei den strukturierten Methoden konnte zwischen zustandsorientierten Verfahren und ereignisfolgenorientierten Verfahren unterschieden werden.

Ereignisfolgenorientierte Verfahren haben den Nachteil, daß unter Umständen sehr viele und sehr lange Ereignisfolgen definiert werden müssen, was nicht nur die Protokollspezifikation unübersichtlich werden läßt, sondern auch beim späteren Protokollablauf eine höhere Reaktionszeit auf ein eingetretenes Ereignis zur Folge hat, da hier anhand einer legalen Ereignisfolge entschieden werden muß, welche Aktionen vom Protokoll auszulösen sind /HÖL87/.

Diesen Nachteil umgehen zustandsorientierte Verfahren, wo das Gedächtnis des Protokolls für vergangene Ereignisse in einem Zustand festgehalten wird. Die Aktionen der Protokoll-einheit können sehr schnell erfolgen, da nur anhand des Zustandes und des eingetretenen Ereignisses entschieden werden muß.

Ereignisfolgenorientierte Verfahren sind zwar für kleine Protokolle oder für Teilaspekte eines Protokolls, wo der Kommunikationsablauf zwischen den beiden Kommunikationspartnern verdeutlicht werden soll, einsetzbar, für komplexere Protokolle und für ein universell einsetzbares Protokollspezifikationsverfahren sind jedoch Methoden, die auf einem zustandsorientierten Verfahren basieren, vorzuziehen.

5.2 Reaktionen auf Ereignisse

Bei zustandsorientierten Verfahren gibt es mehrere Möglichkeiten, wie der Ablauf innerhalb einer Protokolleinheit aussehen kann, wenn ein Ereignis auf das Protokoll trifft. Bild 5.2-1 veranschaulicht die erste dieser Möglichkeiten.

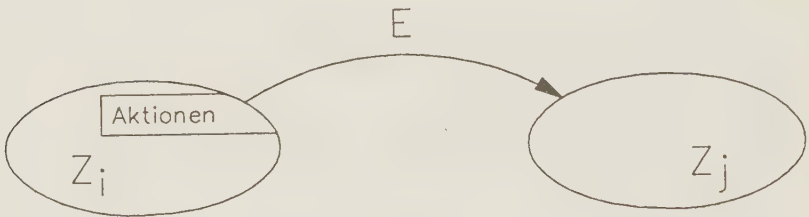


Bild 5.2-1 Auslösen von Aktionen im aktuellen Zustand

Nach Eintreten des Ereignisses werden entsprechend dem Ereignis und dem aktuellen Zustand, in diesem Zustand die Aktionen der Protokolleinheit ausgeführt. Als letzte Aktion wird der Zustandswechsel in den Folgezustand vorgenommen. Bei diesem Ablauf nach Eintreten des Ereignisses sind die Aktionen direkt auf den aktuellen Zustand und das Ereignis bezogen. Der Folgezustand gibt den Status der Protokolleinheit nach Durchführung der Aktionen an, und wie auf folgende Ereignisse zu reagieren ist (vergleichbar mit Mealy Automat).

Eine andere Möglichkeit auf ein Ereignis zu reagieren ist in Bild 5.2-2 gezeigt.

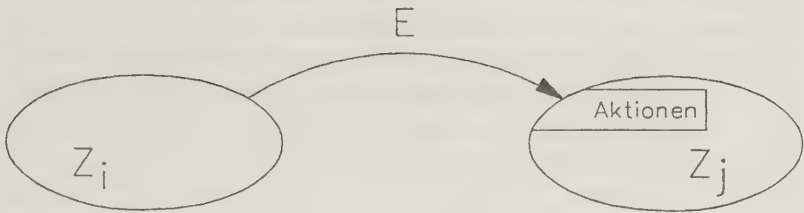


Bild 5.2-2 Auslösen von Aktionen im Folgezustand

Nach Eintreten des Ereignisses wird sofort der Folgezustand eingenommen. In diesem Folgezustand werden die Aktionen der Protokolleinheit ausgeführt. Bei dieser Vorgehensweise bestimmt das eintretende Ereignis nur den Folgezustand. Die Aktionen des Protokolls werden durch den Folgezustand bestimmt. Die Aktionen, die in einem Zustand der Protokolleinheit durchgeführt werden, hängen also nicht mehr direkt von dem Ereignis ab, sondern von dem Zustand selbst (vergleiche Moore Automat).

Eine dritte mögliche Vorgehensweise wäre nun eine Kombination aus den beiden vorgenannten Möglichkeiten. Dies ist in Bild 5.2-3 aufgezeigt.

Nach Eintreten des Ereignisses werden zunächst einmal die Aktionen von der Protokolleinheit durchgeführt, die unmittelbar von dem Ereignis und dem aktuellen Zustand abhängen. Danach wird der Folgezustand eingenommen, und hier werden noch die Aktionen durchgeführt, die rein zustandsabhängig von diesem Folgezustand abhängen.

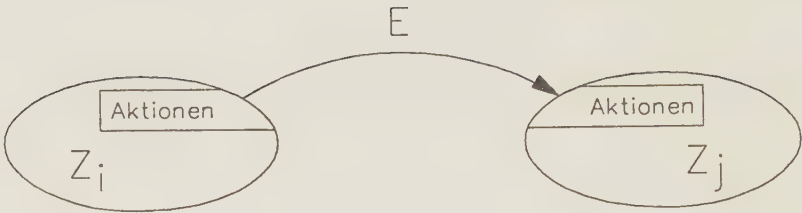


Bild 5.2-3 Aufspalten der Aktionen

Ein Problem für den Designer eines Protokolls stellt die zweite vorgestellte Vorgehensweise dar, da pro Ausgabeaktion der Protokolleinheit ein neuer Zustand definiert werden muß und die Aktionen des Protokolls nicht direkt als Reaktion auf die Ereignisse beschrieben werden. Bei der ersten Möglichkeit den Ablauf darzustellen, ist dagegen der direkte Zusammenhang zwischen eintretendem Ereignis und Protokollaktion gegeben, was das Verständnis für die Funktionsweise der Protokolleinheit, auch bei späterer Implementation durch andere Personen, erleichtert.

Eine Aufspaltung der Protokollaktionen, wie in der dritten möglichen Vorgehensweise beschrieben, würde die gesamte Protokollspezifikation nur verkomplizieren, mit dem einzigen Vorteil, daß eventuell bei verschiedenen Ereignissen, die aus verschiedenen Zuständen in ein und denselben Folgezustand führen, in diesem Zustand für beide Ereignisse gemeinsame Aktionen ausgeführt werden können, falls sich diese gemeinsamen Aktionen überhaupt für alle in diesen Zustand führenden Ereignisse finden lassen. Bei den anderen vorgestellten Vorgehensweisen müßten diese gemeinsamen Aktionen

dann bei den unterschiedlichen Ereignissen getrennt definiert werden.

Die Wahl für den Ablauf innerhalb des Protokolls nach Eintreten eines Ereignisses fiel deshalb auf die zuerst beschriebene Vorgehensweise, wo sämtliche Aktionen der Protokolleinheit im aktuellen Zustand durchgeführt werden, und die letzte Aktion der Zustandswechsel ist.

5.3 Prinzipielle Darstellungsmöglichkeiten

Die einfachste Form ein zustandsorientiertes Verfahren darzustellen, ist ein Automat.

Ein Automat wird definiert als ein Quintupel

$$A = (Z, E, A, f, g)$$

mit der Eingabemenge E , die die Eingabezeichen $e_i \in E$ enthält, der Ausgabemenge A , mit den Ausgabezeichen $a_i \in A$ und der Zustandsmenge Z mit den Zuständen $z_i \in Z$.

Ein Automat befindet sich entweder in einem Zustand z_i oder geht von einem Zustand z_i in einen Folgezustand z_j über, mit $z_i, z_j \in Z$.

f und g sind zwei Abbildungen mit

$$\begin{array}{ll} f: Z \times E \rightarrow Z & \text{Überföhrungsfunktion} \\ g: Z \times E \rightarrow A & \text{Ausgabefunktion} \end{array}$$

Die Überföhrungsfunktion und die Ausgabefunktion werden in Form einer Wertetabelle angegeben. Bild 5.3-1 veranschaulicht die prinzipielle Arbeitsweise eines Automaten und den Ablauf nach Eintreffen eines Ereignisses.

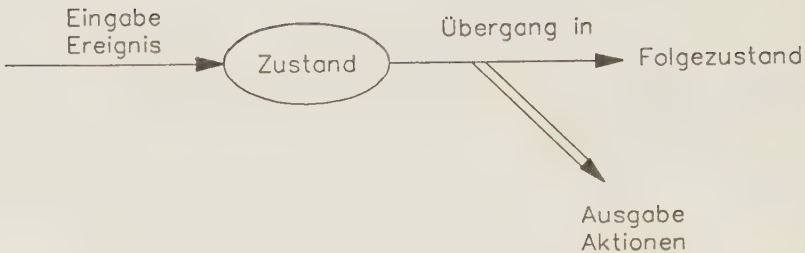


Bild 5.3-1 Ablauf nach Eintreten eines Ereignisses

Ein Ereignis trifft auf einen Automaten, der sich in einem bestimmten Zustand befindet. Entsprechend diesem Zustand werden Ausgaben vorgenommen und ein Folgezustand eingenommen.

Bei der Rechnerkommunikation bedeuten Ein- und Ausgabe keine stetigen Signale, sondern Eingabe für ein Protokoll ist immer das Eintreten eines Ereignisses, wie z. B. der Empfang eines Datenblocks. Ausgabe eines Protokolls bezieht sich immer auf eine Ausgabeaktion an einer Schnittstelle (SAP), wie beispielsweise das Übergeben eines Datenblocks an die nächst tiefere Ebene.

Die Nachteile eines reinen Automaten wurden in einem früheren Abschnitt dieser Arbeit schon aufgezeigt. Größter Mangel war, daß es bei Protokollen sehr schnell zu sogenannten Zustandsexplosionen kommt, wenn auf die Einführung von zusätzlichen Zustandsvariablen verzichtet wird.

Eine andere Schwierigkeit liegt darin, daß die Eingabeereignisse bei Protokollen aus einem komplexen Datenblock bestehen, und die Aktionen und Ausgaben einer Protokoll-einheit sehr umfangreich sein können. Diese komplexen Ein- und Ausgaben lassen sich bei reinen Automaten nur schwer formal beschreiben.

Die Einführung von Zustandsvariablen vermeidet eine Zustandsexplosion, da diese Variablen es erlauben, mehrere ähnliche Zustände zusammenzufassen, wobei dieser eine Zustand dann durch diese Variablen näher spezifiziert wird. Die Aktionen des Protokolls auf einwirkende Ereignisse sind jetzt nicht mehr nur von dem aktuellen Zustand, sondern auch von dem Status der Zustandsvariablen abhängig. Dies wirft jedoch das neue Problem auf, wie die Verknüpfung von eingetretenem Ereignis und den Zustandsvariablen zu realisieren ist.

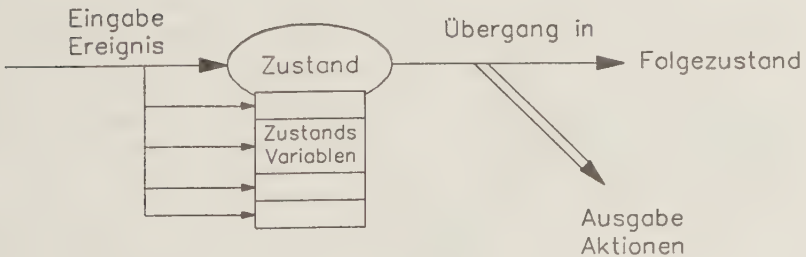


Bild 5.3-2 Verknüpfung mit Zustandsvariablen

Diese Vorgehensweise hat den Nachteil, daß die Protokollspezifikation sehr umfangreich wird, da unter Umständen in jedem Zustand für jedes Ereignis die Verknüpfungsvorschriften neu angegeben werden müssen, wobei es durchaus vorkommt, daß diese Vorschriften für ein bestimmtes Ereignis in unterschiedlichen Zuständen gleich sind.

Aus diesem Grund wurde eine neue Möglichkeit erarbeitet, die eine Vorverarbeitung der auf die Protokolleinheit wirkenden Ereignisse vorsieht. Die Verknüpfung der Ereignisse mit den Zustandsvariablen erfolgt unabhängig von dem jeweiligen Zustand. Als Ergebnis dieser Verknüpfungen entstehen neue Ereignisse, die nun auf die Zustandsmaschine wirken.

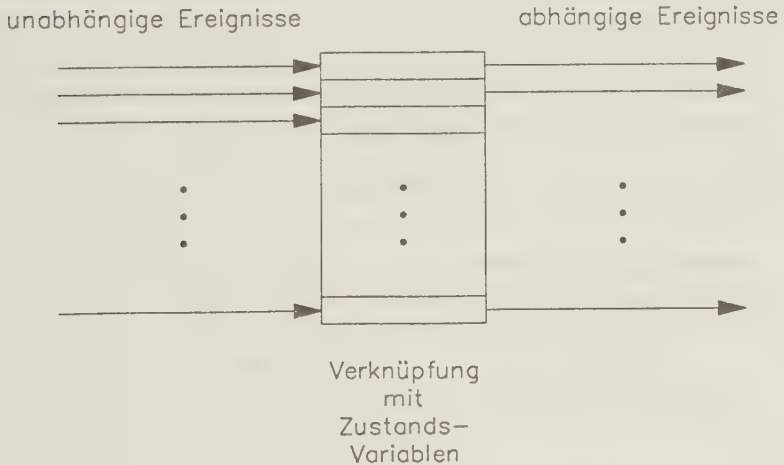


Bild 5.3-4 Vorverarbeitung von Ereignissen

Die Ereignisse, die ursprünglich auf das Protokoll einwirken, werden im folgenden als "unabhängige Ereignisse" bezeichnet, und die Ereignisse, die nach der Verknüpfung der unabhängigen Ereignisse mit den Zustandsvariablen entstehen, als "abhängige Ereignisse". Dieser Zusammenhang ist in Bild 5.3-4 dargestellt.

Auf die Zustandsmaschine wirken jetzt also nur die generierten abhängigen Ereignisse, wobei diese auch bestimmen, welche Aktionen von der Protokolleinheit vorgenommen werden, und welcher Folgezustand eingenommen wird. In Bild 5.3-5 ist diese Vorgehensweise dargestellt.

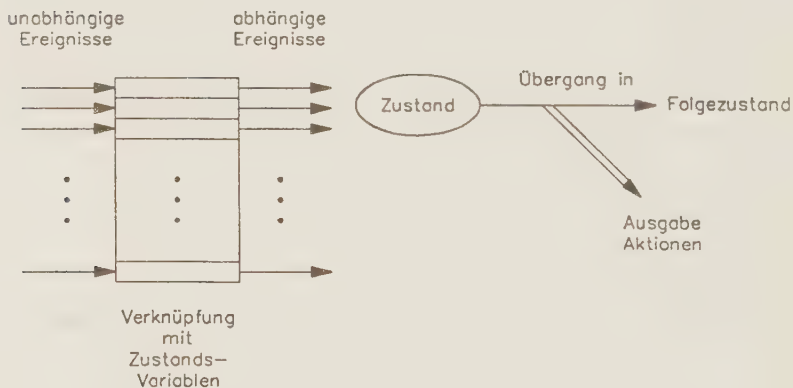


Bild 5.3-5 Ablauf mit Vorverarbeitung von Ereignissen

Die Verknüpfungen mit den Zustandsvariablen werden pro unabhängiges Ereignis nur noch einmal spezifiziert, woraus

sich dann die abhängigen Ereignisse ablesen lassen. Die Definition der Zustandsmaschine erfolgt anhand der abhängigen Ereignisse, wobei jetzt keine Verknüpfungen mehr zu berücksichtigen sind.

Durch diese Aufteilung in zwei Schritte wird die gesamte Protokollspezifikation überschaubarer, und es ist keine Mehrfachbeschreibung von Verknüpfungen mehr nötig.

5.4 Repräsentation der Protokollspezifikation

Ein Hauptaugenmerk bei dem Protokollbeschreibungsverfahren gilt der Darstellung der Protokollspezifikation. Diese ist, neben der Struktur des Verfahrens, dafür verantwortlich, daß die Spezifikation eindeutig und leicht verständlich ist.

Eine verbale Beschreibung in natürlicher Sprache ist, wie früher in dieser Arbeit schon aufgezeigt, nicht immer eindeutig und zu ungenau, um komplexe Vorgänge klar darzustellen. Die verbale Form ist zudem nicht von einem Rechner verarbeitbar, so daß keine Hilfen zur Simulation und zur Dokumentation gegeben werden können.

Eine graphische Repräsentation scheidet ebenfalls aus, da nicht alle Informationen in der Zeichnung untergebracht werden können, und hier auf zusätzliche Beschreibungsformen zurückgegriffen werden müßte.

Programmiersprachenähnliche Verfahren sind im Prinzip bereits eine Implementation des Protokolls und sollen deshalb nicht berücksichtigt werden.

Die Wahl zur Repräsentation der Protokollbeschreibungsmethode fiel deshalb auf eine tabellarische Darstellung in Listenform. Diese Listen haben den Vorteil, daß sie leicht von einer Rechneranlage zu lesen und zu interpretieren sind. Sie ergeben eine exakte und eindeutige Protokollspezifikation, und die Spezifikation ist von einem Rechner simu-

lierbar.

Die Listen können direkt an einem Rechner erstellt werden, wobei eine Benutzerführung durch das Eingabeprogramm vorgesehen werden kann. Damit erfolgt die komplette Spezifikationseingabe rechnerunterstützt. Die Dokumentationserstellung kann somit ebenfalls durch den Rechner vorgenommen werden. Modifikationen der Spezifikation sind schnell und einfach durchführbar, und eine erste Überprüfung auf sinnvolle Anweisungen kann schon während der Erstellung der Spezifikation vorgenommen werden. Da der Benutzer über Menues durch die einzelnen Spezifikationspunkte geführt wird, und jederzeit eine Hilfestellung angeboten wird, ist die Einarbeitungszeit in dieses Protokollentwurfsverfahren sehr klein. Die Listen sind selbst bei komplexen Protokollen noch überschaubar, und da sie keine umgangssprachlichen Texte enthalten, die in andere Sprachen übersetzt werden müßten, für internationale Normen gut geeignet.

Die Struktur der einzelnen Listen sowie der Zusammenhang zwischen Protokolldefinition und späterem Protokollablauf werden im folgenden Kapitel dargestellt.

6 Feinstruktur des Protokollentwurfsverfahrens

In diesem Abschnitt wird die Struktur und der Aufbau der einzelnen Listen, aus denen sich die gesamte Protokollspezifikation zusammensetzt, dargestellt. Dabei wird gleichzeitig der Bezug zwischen späterem Ablauf innerhalb der Protokolleinheit und diesen Listen verdeutlicht.

Den einzelnen Listen ist später eine kurze Beschreibung vorangestellt, aus der die Struktur der Listen ersichtlich ist. Die gesamte Protokollspezifikation ist durchnummeriert, wobei am Kopf jeder Seite folgende Zeile erscheint:

Protokollname Seite uuu / vvv Liste xx Seite yyy / zzz

durchnummeriert.

vvv gibt die gesamte Anzahl der Seiten der Protokollspezifikation an

uuu gibt die aktuelle Seite der gesamten Protokollspezifikation an

xx gibt die Nummer (01 - 19) der gerade aktuellen Liste an

zzz gibt an, wieviele Seiten zur aktuellen Liste gehören

yyy gibt die aktuelle Seite innerhalb der Liste an

Damit kann leicht überprüft werden, ob Seiten der Protokolldokumentation fehlen.

Im folgenden werden häufig Zeichen zum Numerieren oder als Platzhalter für Zahlen benutzt. Dabei sollen folgende Vereinbarungen gelten:

xx, yy, zz - bedeutet Hexadezimalzahl von 00 bis FF
n, m - bedeutet Hexadezimalzahl von 0 bis F
i, k, l - bedeutet Integer Zahl

Die gesamte Protokollspezifikation teilt sich in die folgenden 19 Listen auf:

- 1 Liste der Zustände
- 2 Liste der Konstanten
- 3 Liste der Variablen
- 4 Liste der Felder
- 5 Liste der Common-Felder
- 6 Liste der Datenblöcke
- 7 Liste der unabhängigen Ereignisse
- 8 Liste der Schnittstellenaktionen
- 9 Verknüpfung von Datenblockelementen zu unabhängigen Ereignissen
- 10 Zuweisung von Datenblockelementen bei Schnittstellenaktionen
- 11 Liste der Verknüpfungen nach abhängigen Ereignissen ausgerichtet
- 12 Liste der Verknüpfungen nach unabhängigen Ereignissen ausgerichtet
- 13 Liste der abhängigen Ereignisse
- 14 Liste der Zustandsübergänge
- 15 Liste der zustandsunabhängigen Aktionen

- 16 Liste der zustandsabhängigen Aktionen
- 17 Liste der Aktionen der Funktionen
- 18 Liste der Funktionen
- 19 Definition der Unterzustandsmaschinen

Die einzelnen Elemente der Spezifikation, wie Zustände, Variablen, Aktionen, etc., werden prinzipiell in folgender Weise gekennzeichnet:

Zuerst wird ein Großbuchstabe angegeben, der den Typ des entsprechenden Elements wiedergibt. Diesem folgt ein Unterstrich und eine Durchnumerierung. Die Durchnumerierung erfolgt durch zwei hexadezimale Ziffern von 00 - FF.

6.1 Definition der Zustände der Protokolleinheit

In der Liste 1 werden Zustände, die die Protokolleinheit annehmen kann, aufgeführt. Die Zustände werden mit dem Buchstaben Z gekennzeichnet, gefolgt von Unterstrich und Durchnumerierung, wie oben beschrieben.

Zusätzlich zu dieser formalen Kennzeichnung eines Zustandes wird für jeden Zustand noch ein Name angegeben. Eine optimale Bemerkung soll die Funktion des Zustandes näher erläutern.

Die Form der Liste 1, Liste der Zustände, ist in Bild 6.1-1 dargestellt.

Nr.	Name des Zustandes	Bemerkungen
Z_xx	Zustandsname	erklärender Text
.	..	...
.	..	...

Bild 6.1-1 Liste der Zustände

6.2 Definition von Konstanten und Variablen

Die Definition von Konstanten und Variablen für die Protokolleinheit ist aufgeteilt auf die vier Listen 2-5 der Protokollspezifikation.

Den Konstanten wird in der Spezifikation ein Integer-Wert zugewiesen. Sie sollen immer benutzt werden, wenn in der Protokollspezifikation Vergleiche auf bestimmte Werte durchgeführt oder Variablen initialisiert werden. Dadurch sind Protokolländerungen sehr schnell durchführbar. Wenn z. B. die maximal mögliche Anzahl von Blockwiederholungen oder ein Startwert für einen Timer geändert werden soll, so braucht dies dann nur in der Liste der Konstanten zu erfolgen, ohne daß die gesamte Protokollspezifikation nach diesen Werten durchsucht werden müßte. Die formale Konstantenbezeichnung wird durch einen Namen und eine optionale Bemerkung ergänzt. Die Form der Liste 2, Liste der Konstanten, ist aus Bild 6.2-1 ersichtlich.

Nr.	Konstante	Wert	Bemerkungen
K_xx	Konstantenname	i	erklärender Text
.	..	.	...
.	..	.	...

Bild 6.2-1 Liste der Konstanten

Die Variablen der Protokolleinheit werden mit dem Buchstaben V gekennzeichnet. Sie beinhalten sowohl die Zustandsvariablen des Protokolls als auch ganz normale Variablen. Wird z. B. in einer Variablen die Netzwerkadresse der Station abgelegt, so muß diese beim Versenden eines Datenblocks als Absenderadresse meist mit im Datenblock gesendet werden. In diesem Fall wird die Variable ganz normal und nicht als Zustandsvariable benutzt. Beim Empfang eines Datenblocks hingegen wird ein Vergleich zwischen der Netzwerkadresse der Station und der Zieladresse im Datenblock nötig sein, um daraus, falls der Datenblock für die Station bestimmt ist, ein Ereignis zu generieren. Jetzt wird die Variable, die die Netzwerkadresse beinhaltet, also als Zustandsvariable verwendet. Eine klare Trennung zwischen Variablen und Zustandsvariablen ist also nicht in allen Fällen möglich. Aus diesem Grund werden auch alle Variablen in einer Liste spezifiziert, und es wird keine Trennung zwischen Zustandsvariablen und normalen Variablen vorgenommen.

Die Variablenbezeichnung wird wieder durch einen Variablennamen und eine Bemerkung ergänzt. Der Wertebereich der Variablen wird durch eine untere und eine obere Grenze angegeben, wobei hier Zahlen, jedoch vorzugsweise Konstanten benutzt werden können.

Bei großen Zahlen ist es möglich, den Wert in einer exponentiellen Schreibweise anzugeben. Werden für eine Adresse 48

Bit reserviert, so kann in der Variablendefinition die untere Grenze des Wertebereichs mit 0 und die obere Grenze mit $2^{**48} - 1$ angegeben werden.

Für logische Variablen gilt die Vereinbarung, daß der Wert 0 als FALSE betrachtet wird und alle übrigen Werte als TRUE. Diese Vereinbarung gilt auch bei den später beschriebenen Verknüpfungen mit den Zustandsvariablen.

Die Form der Liste 3, Liste der Variablen, ist in Bild 6.2-2 dargestellt.

Nr.	Name	von	bis	Bemerkungen
V_xx	Variablenname	i	k	erklärender Text
V_xx	Variablenname	i	K_xx	erklärender Text
V_xx	Variablenname	K_xx	i	erklärender Text
V_xx	Variablenname	K_xx	K_yy	erklärender Text
.	..	.	.	...
.	..	.	.	...

Bild 6.2-2 Liste der Variablen

In manchen Fällen ist es nötig, Zustandsvariablen als Felder zu organisieren. Dies ist z. B. beim Token-Passing Buszugriffsverfahren /802.485/, wo die "token rotation time" von der gerade aktuellen "access-class" abhängt, der Fall /SÖN86/. Gleichzeitig werden auch ganz normale Datenfelder benötigt, in denen z. B. zu versendende Daten abgelegt sind. Diese Felder des Protokolls werden in der Liste 4 definiert.

Felder werden mit dem Buchstaben F gekennzeichnet. Die Größe des Feldes wird über eine Variable angegeben, wobei der Wertebereich der definierten Variablen die Anzahl der Feldelemente angibt. Der Wertebereich der einzelnen Feldelemente wird, wie schon bei der Variablendefinition in Liste 3, durch eine untere und eine obere Grenze spezifiziert. Die Liste 4 der Felder hat somit die in Bild 6.2-3 dargestellte Form.

Nr.	Name	von	bis	Bemerkungen
F_xx(V_yy)	Feldname	i	K_xx	erklärender Text
..	..	.	.	...
..	..	.	.	...

Bild 6.2-3 Liste der Felder

Beim späteren Zugriff auf Feldelemente sind Zahlen, Konstanten, Variablen und Feldelemente erlaubt. Wird auf ein Feldelement über den Wert eines anderen Feldelements zugegriffen, so sind bei der Spezifikation des zweiten Feldelements nur noch Zahlen, Konstanten und Variablen zugelassen.

Diese Einschränkung wurde gemacht, damit sich die späteren Vergleichsoperationen und Aktionen der Protokolleinheit noch vernünftig in eine Listenform bringen lassen. Die Erfahrung im praktischen Einsatz hat jedoch gezeigt, daß dies in keiner Weise zu einer Beeinträchtigung bei der Protokollspezifikation führt /SÖN86/RAD88/JÄG86/VER87/.

Die letzte Liste, in der Variablen definiert werden können, ist die Liste 5, Liste der Common-Felder. Die Definition der Common-Felder ist praktisch gleich mit der Definition der Felder aus Liste 4, jedoch erstreckt sich ihre Gültigkeit auf alle zur Protokollspezifikation gehörenden Unterzustandsmaschinen. Sie werden in der Hauptzustandsmaschine definiert und sind in allen Unterstrukturen gleich. Auf eine nähere Beschreibung dieses Zusammenhangs wird am Ende dieses Kapitels nochmals eingegangen, nachdem die Funktion und der Einsatz von Unterzustandsmaschinen erläutert wurden. Common-Felder werden mit dem Buchstaben C gekennzeichnet. Die Form der Liste 5 ist in Bild 6.2-4 dargestellt.

Nr.	Name	von	bis	Bemerkungen
C_xx(V_yy)	Feldname	i	K_xx	erklärender Text
..	..	.	.	...
..	..	.	.	...

Bild 6.2-4 Liste der Common Felder

6.3 Definition der Datenblöcke für die Protokolleinheit

Wie in Kapitel 2.3. schon erwähnt, erfolgt der Datenaustausch zwischen Protokollebenen über sogenannte SAPs (Service Access Points). Über diese SAPs können Datenblöcke eine Protokolleinheit erreichen, und das Senden eines Datenblocks an eine Nachbarebene erfolgt über deren SAP. Es gibt also an jeder Ebenengrenze einen SAP, über den die beiden

Protokollebenen Datenblöcke austauschen (vgl. Bild 6.3-1).

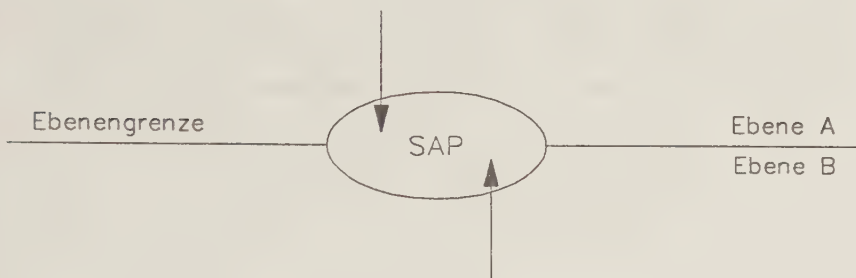


Bild 6.3-1 Kommunikation über SAP

Da die Datenblöcke, die von der Ebene A zur Ebene B gesendet werden, nicht mit denen identisch sein müssen, die von der Ebene B zur Ebene A gesendet werden, wird eine Aufteilung des SAPs für die beiden Datenflußrichtungen vorgenommen (vergleiche Bild 6.3-2).

Damit ergibt sich für die Protokolleinheit einer Ebene am SAP ein Datenblockformat für Datenblöcke, die von der Protokolleinheit empfangen werden, und ein Datenblockformat für Datenblöcke, die von dem Protokoll gesendet werden. Dies gilt sowohl für den SAP zur nächst höheren Ebene, wie auch für den SAP zur nächst tieferen Ebene.

Eine Protokolleinheit kann natürlich über mehr als nur die beiden SAPs zu den Nachbarebenen verfügen. Dies ist z. B. bei Rechnern der Fall, auf denen Protokolle implementiert sind, die eine Konvertierung unterschiedlicher Netze vornehmen (Gateways, Bridges).

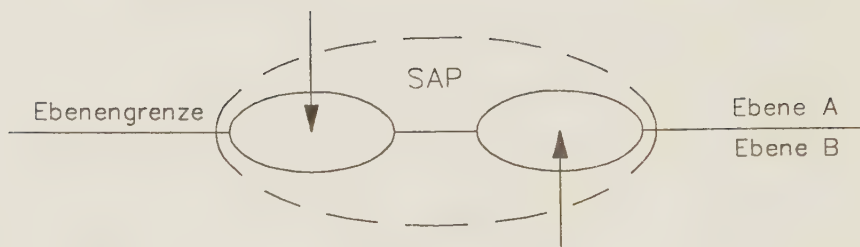


Bild 6.3-2 Aufspaltung des SAP

Ein solcher Zusammenhang ist in Bild 6.3-3 für ein Protokoll, das bei gleichen Protokollen in den höheren Ebenen eine Brückenfunktion zwischen zwei unterschiedlichen Netzen (z. B. Token Bus - CSMA/CD) realisiert, dargestellt.

Bei diesem Protokollspezifikationsverfahren sind pro Protokolleinheit bis zu 16 SAPs zugelassen. Dies erscheint bei weitem ausreichend, wenn für ein Gateway möglicherweise 3 SAPs zu unterschiedlichen Protokolleinheiten der tieferen Ebene, 2 SAPs zur höheren Ebene und möglicherweise ein SAP für den Zugriff eines "Station Management" vorgesehen werden müßten.

Datenblöcke werden mit dem Buchstaben D gekennzeichnet. Für die Durchnummerierung der Datenblöcke gilt folgende Vereinbarung:

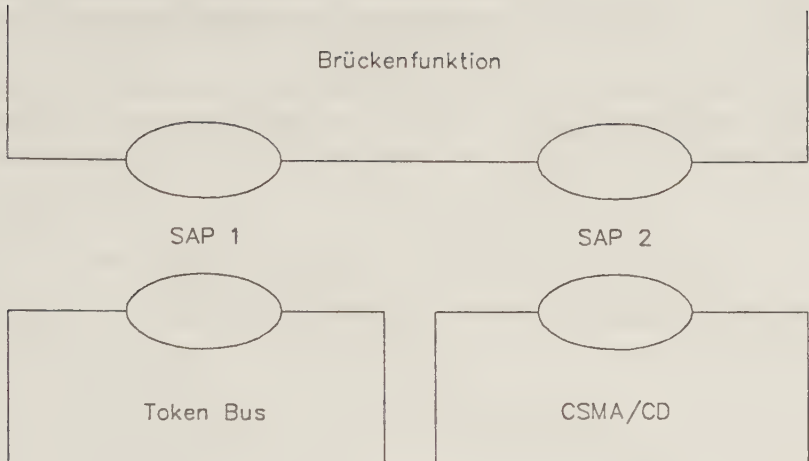


Bild 6.3-3 Protokolleinheit mit Brückenfunktion

D_0n Bezeichnet den Datenblock, der von einer Nachbar-
ebene am SAP n an die Protokolleinheit gesendet
wird

D_1n Bezeichnet den Datenblock, der von der Protokoll-
einheit am SAP n an eine Nachbarebene gesendet
wird.

n bezeichnet jeweils die Nummer des SAP und kann
eine Hexadezimalziffer von 0 - F sein.

Die Größe des Datenblocks wird, wie bei der Definition der
Felder, durch eine Variable spezifiziert, wobei der Werte-
bereich der Variablen wieder die Anzahl der Datenblockele-

mente angibt. Der Wertebereich der einzelnen Datenblockelemente wird durch eine untere und eine obere Grenze festgelegt.

Die Form der Liste 6, Liste der Datenblöcke, ist in Bild 6.3-4 dargestellt.

Nr.	Name	von	bis	Bemerkungen
D_On(V_xx)	Block-Name	i	K_xx	Anford. an Ebene
..	..	.	.	...
..	..	.	.	...

Bild 6.3-4 Liste der Datenblöcke

Die Datenblockdefinitionen der Protokolleinheit stehen natürlich im direkten Zusammenhang mit den Datenblockdefinitionen der Protokolle der Nachbarebenen, da ein über einen SAP an eine Nachbarebene versandter Datenblock mit dem empfangenen Datenblock der Nachbarebene identisch ist.

Dies gilt ebenso für die empfangenen Datenblöcke, die von dem Protokoll einer Nachbarebene versendet werden.

Es besteht hier also schon eine erste formelle Möglichkeit zu überprüfen, ob die Datenblockformate an einem SAP für die beiden beteiligten Protokolle gleich sind (jeweils in Send- und Empfangsrichtung).

6.4 Definition der auf das Protokoll wirkenden Ereignisse und der Schnittstellenaktionen

Wie in Abschnitt 3.2. dieser Arbeit schon aufgezeigt, gibt es verschiedene Ereignistypen, die auf die Protokolleinheit einwirken können. Diese sind in Bild 6.4-1 nochmals kurz dargestellt.

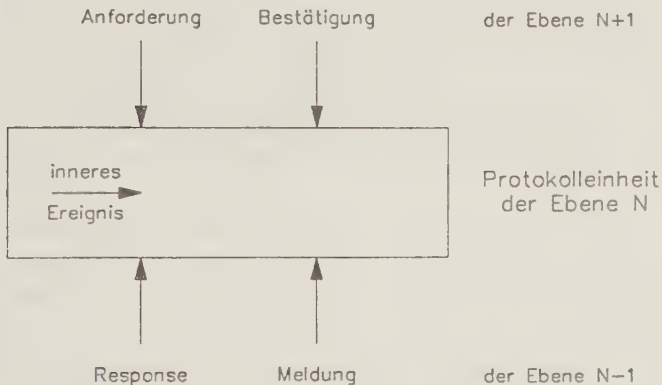


Bild 6.4-1 Auf das Protokoll einwirkende Ereignisse

Bis auf die inneren Ereignisse, die im allgemeinen durch abgelaufene Timer ausgelöst werden, treffen alle Ereignisse über die Service Access Points auf die Protokolleinheit. Dabei entsprechen die Anforderungen und Bestätigungen den Schnittstellenaktionen der höheren Ebene, und die Responses und Meldungen den Schnittstellenaktionen der tieferen Ebene. Die ursprünglich auf das Protokoll einwirkenden Ereignisse, die, wie in Abschnitt 5.2 dargestellt, als unabhängige Er-

eignisse bezeichnet werden sollen, entsprechen also direkt den Aktionen oder Ausgaben der Nachbarebenen. Ebenso werden die Schnittstellenaktionen der Protokolleinheit zu unabhängigen Ereignissen in den Nachbarebenen.

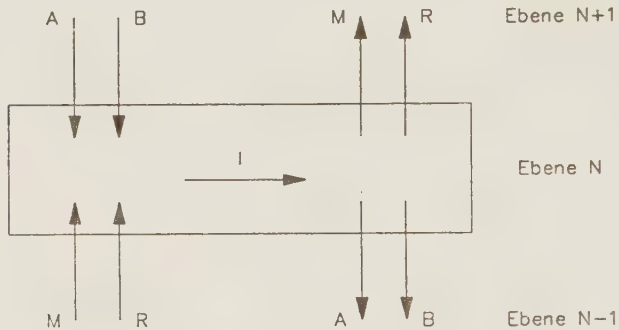


Bild 6.4-2 Ereignisse und Schnittstellenaktionen

Dieser Zusammenhang ist in Bild 6.4-2 dargestellt, wobei ein Pfeil in Richtung einer Ebene ein unabhängiges Ereignis für diese Ebene bedeutet, und ein Pfeil, der von einer Ebene wegzeigt, einer Schnittstellenaktion in dieser Ebene entsprechen soll.

Dieser Mechanismus gilt an jedem SAP der Protokolleinheit, weshalb bei jedem unabhängigen Ereignis und jeder Schnittstellenaktion mit angegeben werden muß, an welchem SAP sie eintreffen bzw. ausgelöst werden sollen.

Die unabhängigen Ereignisse werden deshalb folgendermaßen bezeichnet:

I_xx	Inneres Ereignis (abgelaufener Timer) der Protokolleinheit
An_xx	Anforderung an die Protokolleinheit am SAP n von der Nachbarebene
Bn_xx	Bestätigung an die Protokolleinheit am SAP n von der Nachbarebene
Mn_xx	Meldung an die Protokolleinheit am SAP n von der Nachbarebene
Rn_xx	Antwort (Response) an die Protokolleinheit am SAP n von der Nachbarebene

Dabei wird die SAP Nummer n durch eine Hexadezimalziffer von 0-F für die SAPs 0-15 angegeben.

Die Form der Liste 7, Liste der unabhängigen Ereignisse, ist in Bild 6.4-3 dargestellt.

Nr.	Name des unabh. Ereignisses	Bemerkungen
An_xx	Name der Anforderung	erklärender Text
Bn_xx	Name der Bestätigung	erklärender Text
Mn_xx	Name der Meldung	erklärender Text
Rn_xx	Name der Antwort	erklärender Text
I_xx	Name des Timers	erklärender Text
.	..	...
.	..	...

Bild 6.4-3 Liste der unabhängigen Ereignisse

Bei der Erstellung dieser Liste können die Ereignisse für alle SAPs in einer Liste eingegeben werden. Bei der automa-

tischen Dokumentationserstellung wird jeweils eine eigene Seite für jeden benutzten SAP ausgegeben.

Die Schnittstellenaktionen der Protokolleinheit werden ganz ähnlich wie die unabhängigen Ereignisse angegeben und zwar bedeuten:

An_xx	Anforderung der Protokolleinheit am SAP n an die Nachbarebene
Bn_xx	Bestätigung der Protokolleinheit am SAP n an die Nachbarebene
Mn_xx	Meldung der Protokolleinheit am SAP n an die Nachbarebene
Rn_xx	Antwort (Response) der Protokolleinheit am SAP n an die Nachbarebene
n	bezeichnet wieder jeweils die Nummer des SAP, an dem die Schnittstellenaktion ausgelöst werden soll.

Die gleiche Bezeichnung von Ereignissen und Aktionen führt zu keinerlei Konflikten innerhalb der Protokollspezifikation, da immer ganz klar zwischen unabhängigem Ereignis und Schnittstellenaktion unterschieden werden kann.

Die Form der Liste 8, Liste der Schnittstellenaktionen, ist in Bild 6.4-4 angegeben.

Auch bei dieser Liste werden die Aktionen bei der Dokumentationserstellung nach SAPs sortiert und pro SAP eine neue Liste erstellt.

Nr.	Name der Schnittstellenakt.	Bemerkungen
An_xx	Name der Anforderung	erklärender Text
Bn_xx	Name der Bestätigung	erklärender Text
Mn_xx	Name der Meldung	erklärender Text
Rn_xx	Name der Antwort	erklärender Text
.	..	...
.	..	...

Bild 6.4-4 Liste der Schnittstellenaktionen

6.5 Zusammenhang zwischen den Datenblöcken und den unabhängigen Ereignissen und den Schnittstellenaktionen

Sämtliche Kommunikation zwischen den Protokollebenen erfolgt in Form von Datenblöcken, die über die Service Access Points der Ebenen weitergeleitet werden. Daraus ergibt sich, daß auch die Ereignisse für die Protokolleinheit und die Schnittstellenaktionen in diesen Datenblöcken verschlüsselt sein müssen. Die einzige Ausnahme bilden die inneren Ereignisse, da diese durch abgelaufene Timer direkt auf das Protokoll wirken und nicht in Form eines Datenblocks an einem SAP.

6.5.1 Erkennen von unabhängigen Ereignissen

Das eingetretene Ereignis muß also zunächst aus dem Datenblock gelesen werden. Es gibt einen direkten Zusammenhang zwischen bestimmten Datenblockelementen und den auf das Protokoll einwirkenden unabhängigen Ereignissen. Je nach Inhalt dieser Datenblockelemente werden entsprechend unterschiedliche Ereignisse generiert.

Der Ablauf innerhalb des Protokolls ist in Bild 6.5.1-1 dargestellt.

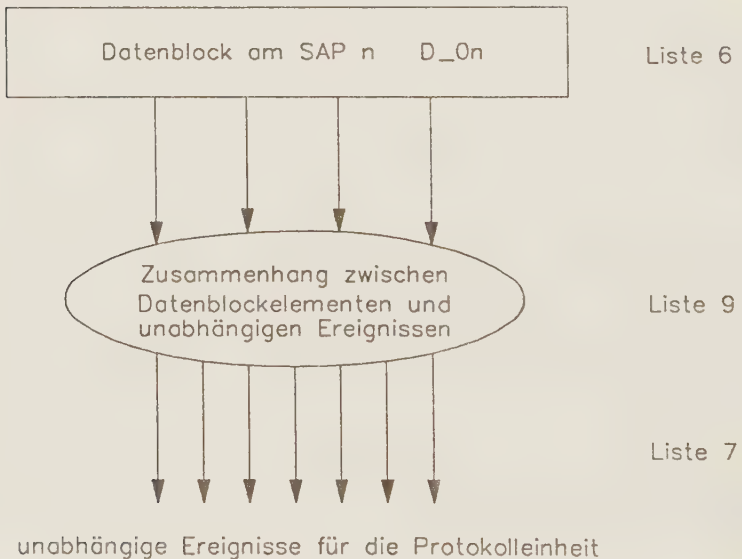


Bild 6.5.1-1 Erzeugen der unabhängigen Ereignisse

Ein Datenblock (definiert in Liste 6) erreicht die Protokolleinheit an einem SAP. Je nach Inhalt bestimmter Datenblockelemente werden unabhängige Ereignisse für das Protokoll generiert, die in Liste 7 beschrieben sind. Dabei sind die verschiedensten Verknüpfungsoperationen zugelassen. Die Verknüpfung von Datenblockelementen zu unabhängigen Ereignissen ist in Liste 9 beschrieben. Die Form dieser Liste 9 ist in Bild 6.5.1-2 dargestellt.

	Xn_xx	Xn_yy
D_0n(i)	== K_yy	>= V_zz
D_0n(k)	/= 0	== V_yy & K_zz
.	.	.
.	.	.

Bild 6.5.1-2 Liste der Verknüpfungen

In der ersten Spalte dieser Liste stehen die Datenblockelemente, die für die Verknüpfungen benötigt werden. Über den folgenden Spalten steht jeweils das unabhängige Ereignis (hier durch den Buchstaben X gekennzeichnet), das ausgelöst wird, wenn die AND - Verknüpfungen der Terme innerhalb der Spalte den Wert TRUE ergeben. Dazu können die folgenden Vergleichsoperationen angewendet werden.

- == Das Datenblockelement ist gleich dem angegebenen Term
- /= ungleich
- > größer als

>= größer gleich
< kleiner
<= kleiner gleich

Neben diesen Vergleichsoperatoren zwischen Datenblockelement und Term sind noch Operatoren innerhalb der Terme zugelassen. Diese beinhalten zunächst einmal die vier Grundrechenarten und werden bezeichnet mit

+ Addition
- Subtraktion
** Multiplikation
// Division

Weiterhin stehen Operatoren zur Verfügung, die einzelne Bits innerhalb der angegebenen Daten manipulieren, und zwar

/ Negation (Zweierkomplement)
& bitweises logisches AND
/& bitweises logisches NAND
\$ bitweises logisches OR
/\$ bitweises logisches NOR

Zusätzlich können noch die folgenden Rotationsbefehle

<- Rotation links
-> Rotation rechts

angewendet werden, wobei jeweils das an einer Seite des Datenwortes herausgeschobene Bit am anderen Ende wieder hineingeschoben wird.

Die Operationen wirken grundsätzlich auf den links von ihnen stehenden Term, das heißt die Zeile

D_0n(i) == V_xx & K_yy

vergleicht das Datenblockelement mit der Variablen V_xx, die mit der Konstanten K_yy AND - verknüpft ist, und die Zeile

$$D_On(i) \quad \& \quad K_yy == V_xx$$

vergleicht das mit der Konstanten K_yy AND verknüpfte Datenblockelement mit der Variablen V_xx.

Als Elemente innerhalb der Terme sind in dieser Liste Zahlen, Konstanten, Variablen und Feldelemente zugelassen, wobei der Zugriff auf Feldelemente, wie in Abschnitt 6.2 dargestellt, durchgeführt werden muß.

Der Normalfall für das Erkennen eines unabhängigen Ereignisses im Datenblock ist sicher der, daß ein Datenblockelement für die unterschiedlichen Ereignisse unterschiedliche Werte annimmt, das Ereignis also direkt an einer bestimmten Stelle kodiert im Datenblock steht. Dazu reichen die angegebenen Vergleichsoperationen aus. Nur in einigen speziellen Fällen sind umfangreichere Operationen durchzuführen, die dann durch die angegebenen zusätzlichen Operatoren abgedeckt sind.

6.5.2 Auslösen von Schnittstellenaktionen

Das Auslösen einer Schnittstellenaktion bedeutet für die Protokolleinheit den Aufbau eines Datenblocks und das Weiterleiten dieses Datenblocks an den entsprechenden SAP der Nachbarebene. Dazu müssen die entsprechenden Aktionen, die zu Ereignissen in den Nachbarebenen werden, vorher in den Datenblock eingetragen werden.

Bild 6.5.2-1 zeigt diesen Ablauf. Die Schnittstellenaktionen, die in Liste 8 definiert sind, werden an entsprechende Stellen im Datenblock, definiert in Liste 6, eingetragen.

Den Zusammenhang, wie die Aktionen zu verschlüsseln sind und welche Datenblockelemente welche Werte bei der entsprechenden Schnittstellenaktion erhalten müssen, wird in Liste 10, Zuweisung von Datenblockelementen bei Schnittstellenaktionen, angegeben.

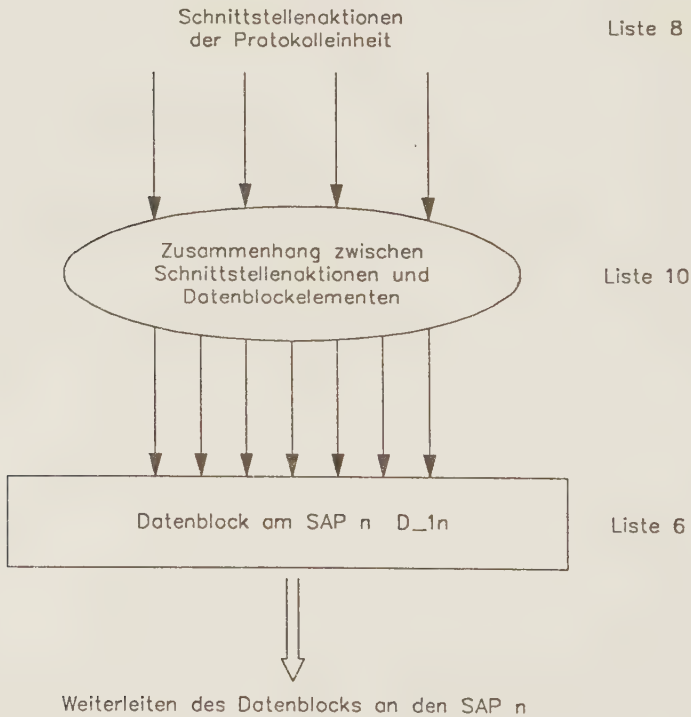


Bild 6.5.2-1 Belegen von Datenblockelementen

Die Form der Liste 10 ist in Bild 6.5.2-2 dargestellt.

	Xn_yy	Xn_zz
D_ln(i)	:= K_xx	
D_ln(k)	:= 15	-> 5
D_ln(l)	=* V_xx,K_yy	
.	.	.
.	.	.

Bild 6.5.2-2 Zuweisung von Datenblockelementen

In der ersten Spalte dieser Liste stehen die Datenblockelemente, denen ein Wert zugewiesen werden soll. Über den folgenden Spalten steht jeweils die Schnittstellenaktion, hier durch den Buchstaben X gekennzeichnet. Beim Auslösen der Schnittstellenaktion werden den Datenblockelementen die in der entsprechenden Spalte stehenden Werte zugewiesen. Anschließend wird der Datenblock an den SAP n weitergeleitet. Für die Zuweisung der Datenblockelemente sind folgende Operatoren definiert:

:= Zuweisung des angegebenen Wertes

Dabei ist es möglich, einem Datenblockelement einen Wert zuzuweisen oder ein Feld oder Teile eines Feldes in den Datenblock ab einer bestimmten Stelle zu kopieren. Für die Kopieroperation muß auf beiden Seiten das Startelement innerhalb des Datenblocks bzw. des Feldes spezifiziert werden. Die Anzahl der zu kopierenden Elemente wird hinter der Anweisung, durch Komma getrennt, angegeben.

Die Anweisung

```
D_ln(i) := F_xx(j),k
```

kopiert k Elemente aus dem Feld F_xx beginnend mit dem Feld-
element j in den Datenblock D_ln ab dem Datenblockelement i.

Neben dieser einfachen Zuweisung gibt es die folgenden
arithmetischen Operatoren:

```
+      Addition  
-      Subtaktion  
**     Multiplikation  
//     Division
```

sowie die logischen Operatoren

```
/      Negation  
&      bitweises logisches AND  
/&     bitweises logisches NAND  
$      bitweises logisches OR  
/$     bitweises logisches NOR
```

und die Rotationsbefehle

```
<-     Rotation links  
->     Rotation rechts
```

wobei das an einem Ende herausgeschobene Bit am anderen Ende
des Wortes wieder eingeschoben wird.

Im Normalfall wird sicher die direkte Zuweisung eines Wertes
zu einem Datenblockelement oder das Kopieren eines Teiles
des Datenblocks ausreichen. Die restlichen Operatoren werden
nur in wenigen speziellen Ausnahmefällen benötigt.

Mit dem Weiterleiten eines Datenblocks ist damit aber immer

auch das Kopieren der Daten, insbesondere der Nutzerdaten, erforderlich. Dieser Vorgang tritt in allen Protokolleinheiten auf. Wurde die Ebenenaufteilung nach dem ISO/OSI Modell gewählt, so sind an der Kommunikation in jedem Kommunikationspartner sieben Protokollebenen beteiligt, womit ein Datenblock, der verschickt werden soll, 14 mal, in jedem Rechner 7 mal, umkopiert werden muß. Dies erfordert einen großen Zeitaufwand in den an der Kommunikation beteiligten Rechnern.

Dieses häufige Umkopieren großer Datenmengen könnte umgangen werden, wenn statt des gesamten Datenblocks nur ein Zeiger auf die Daten über die SAPs weitergeleitet werden muß. Dazu muß allerdings die Protokolleinheit, in der die Daten stehen, diese Daten so lange unverändert lassen, bis sie eine Meldung bekommt, daß diese Daten nicht mehr benötigt werden und der Speicherbereich wieder genutzt werden kann. Wird dieser Mechanismus in allen Ebenen benutzt, so entfällt das komplette Umkopieren der Nutzerdaten, was eine erhebliche Geschwindigkeitssteigerung für den gesamten Datentransfer bedeutet. Das Weiterleiten der Adresse auf die Daten erfordert eine exakte und eindeutige Spezifikation aller Protokollebenen, damit nicht einige Entwickler eines Protokolls doch eine "Sicherheitskopie" des Datenblocks anlegen, weil aus einer nicht ganz eindeutigen Spezifikation nicht hervorgeht, daß der Datenblock wirklich unverändert und vorhanden bleibt, bis der Datentransfer für diese Daten abgewickelt ist.

Die beiden Operatoren, die für diesen Mechanismus zur Verfügung gestellt werden, sind:

- =& Zuweisung der Adresse eines Datenelements
- =* Zuweisung von Daten über eine Adresse.

Die Operanden, auf die die beiden angegebenen Operatoren wirken, sind Felder, Common-Felder oder Datenblöcke. Dabei

muß stets angegeben werden, mit welchem Feldelement beginnend die Operation ausgeführt werden soll.

Die Anweisung

$$D_ln(k) =\& F_xx(K_yy)$$

trägt die Adresse des spezifizierten Feldes in das angegebene Datenblockelement ein, und die Anweisung

$$F_xx(K_yy) =* D_0n(k), i$$

kopiert i Datenelemente ab der Adresse, die in dem Datenblockelement $D_0n(k)$ steht, in das Feld F_xx , beginnend ab dem Feldelement K_yy .

6.6 Vorverarbeitung von unabhängigen Ereignissen

Wie bereits in Abschnitt 5.3 dargestellt, findet eine Vorverarbeitung der auf das Protokoll wirkenden Ereignisse, die als unabhängige Ereignisse bezeichnet werden, statt.

Es werden Verknüpfungen mit den Zustandsvariablen vorgenommen, und je nach Wert dieser Variablen werden entsprechende abhängige Ereignisse erzeugt, die auf die Zustandsmaschine wirken. Bild 6.6-1 zeigt diesen Zusammenhang.

Die Verknüpfungen unabhängiger Ereignisse mit Zustandsvariablen, und die daraus resultierenden abhängigen Ereignisse, werden in der Liste 11, Liste der Verknüpfungen nach abhängigen Ereignissen ausgerichtet, definiert.

Die Liste ist so aufgebaut, daß in der ersten Spalte alle benötigten Variablen, Feldelemente, Datenblockelemente und Common-Feldelemente sowie die relevanten unabhängigen Ereignisse aufgeführt sind. Über der Liste steht das abhängige

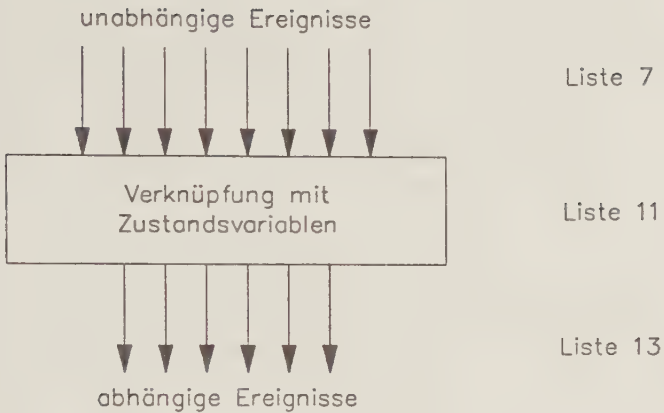


Bild 6.6-1 Erzeugen von abhängigen Ereignissen aus unabhängigen Ereignissen

Ereignis, das generiert wird, wenn das Ergebnis der Verknüpfung, die durch die Liste beschrieben ist, den Wert TRUE ergibt. Die Verknüpfungen in dieser Liste können in zwei alternativen Formen angegeben werden. Einmal eine Liste, deren Terme innerhalb der Spalte AND verknüpft sind und deren Spalten OR verknüpft sind (disjunktive Form). Dies ist in Bild 6.6-2 dargestellt.

E_xx	Ereignisname	erklärender Text
------	--------------	------------------

AND	OR 1
-----	------

Xn_yy	*
V_xx	== K_zz
.	.
.	.

AND	OR 2
-----	------

Xn_yy	*
F_xx(V_yy)	== V_zz /& K_xx
.	.
.	.

Bild 6.6-2 Erzeugen von abhängigen Ereignissen
(disjunktive Form)

Die alternative Form enthält Terme, die innerhalb der Spalte OR verknüpft sind und deren Spalten AND verknüpft sind (konjunktive Form), wie in Bild 6.6-3 dargestellt.

Für beide Formen dieser Liste gilt, daß wenn die Spalten innerhalb der Liste nicht ausreichen, um die Verknüpfungen zu beschreiben, eine neue Liste für das gleiche abhängige Ereignis angehängt wird, wobei die Durchnummerierung der Spalten kontinuierlich fortgesetzt wird. Dies erfolgt jedoch automatisch durch das Programmpaket zum Erstellen der Protokollspezifikation bei der automatischen Dokumentationserstellung.

E_xx	Ereignisname	erklärender Text
------	--------------	------------------

OR	AND 1
----	-------

Xn_yy	*
V_xx	== K_zz
.	.
.	.

OR	AND 2
----	-------

Xn_yy	*
F_xx(V_yy)	== V_zz /& K_xx
.	.
.	.

Bild 6.6-3 Erzeugen von abhängigen Ereignissen
(konjunktive Form)

Innerhalb der Liste 11 sind die folgenden Operatoren definiert:

* unabhängiges Ereignis ist eingetreten

Dieser Operator wird nur auf unabhängige Ereignisse angewendet und kennzeichnet, daß das entsprechende Ereignis eingetreten sein muß.

Weiterhin sind die folgenden Vergleichsoperatoren definiert und zwar:

==	Zustandsvariable ist gleich dem angegebenen Term oder Teile von Feldern sind gleich
/=	ungleich
>	größer
>=	größer gleich
<	kleiner
<=	kleiner gleich

Neben diesen reinen Vergleichsoperatoren stehen noch Operatoren zur Verfügung, die eine Manipulation der Terme zulassen. Dazu gehören die arithmetischen Operatoren:

+	Addition
-	Subtraktion
**	Multiplikation
//	Division

und die logischen Operatoren

/	Negation (Zweierkomplement)
&	bitweises logisches AND
/&	bitweises logisches NAND
\$	bitweises logisches OR
/\$	bitweises logisches NOR

sowie die beiden Rotationsbefehle

<-	Rotation links
->	Rotation rechts

Als Operanden innerhalb der Terme sind Zahlen, Konstanten, Variablen, Feldelemente usw. zugelassen, und die Operatoren wirken grundsätzlich, wie schon in Abschnitt 6.5.1. beschrieben, auf den links von ihnen stehenden Operanden. Zusätzlich zu diesen vorstehenden Operatoren werden noch zwei weitere benötigt, die im Zusammenhang mit der Möglich-

keit stehen, über die SAPs die Adresse eines Datenblocks zu übergeben. Der Operator

=& gleich der Adresse eines Feldes

überprüft, ob die Adresse des spezifizierten Feldes mit der in der ersten Spalte stehenden Variablen o.ä. übereinstimmt, und der Operator

=* gleich dem Inhalt eines Feldes über Adresse

überprüft, ob der in der ersten Spalte angegebene Teil eines Feldes mit dem Datenbereich, dessen Adresse rechts von dem Operator angegeben wird, übereinstimmt.

In der Liste 11 wird für jedes abhängige Ereignis eine eigene Tabelle aufgestellt. Somit ist sehr schnell ersichtlich, auf welche Art und Weise, und aus welchen möglichen unabhängigen Ereignissen, ein abhängiges Ereignis generiert wird, das ja für die Aktionen der Protokolleinheit und den Zustandswechsel verantwortlich ist. Diese Form der Darstellung fördert also die Einarbeitung in das spezifizierte Protokoll, und dient dem leichteren Verständnis des Protokollablaufs.

Für die Implementation der Protokolleinheit wäre jedoch eine andere Form, die nach unabhängigen Ereignissen ausgerichtet ist, wünschenswert. Trifft nämlich ein unabhängiges Ereignis auf die Protokolleinheit, so interessieren nur die Verknüpfungen mit diesem unabhängigen Ereignis, und das abhängige Ereignis, das aus diesen Verknüpfungen entsteht.

Aus diesem Grund wird, automatisch durch das entwickelte Programmpaket zur Protokollspezifikation, eine weitere Liste aus dieser Liste 11 erzeugt, die nach unabhängigen Ereignissen sortiert ist. Diese Liste kann deshalb automatisch generiert werden, da der Inhalt beider Listen vollkommen identisch ist, nur einmal nach abhängigen Ereignissen sortiert und einmal nach unabhängigen.

Die Form dieser generierten Liste 12, Liste der Verknüpfungen nach unabhängigen Ereignissen ausgerichtet, ist in Bild 6.6-4 dargestellt.

X_xx	Ereignisname	erklärender Text
	E_xx	E_yy
V_xx	== K_zz	
V_yy	. .	> V_zz
.	. .	
.	. .	

Bild 6.6-4 Erzeugen von abhängigen Ereignissen
nach unabhängigen Ereignissen ausgerichtet

Über der Liste steht jetzt das unabhängige Ereignis, in der ersten Spalte die Zustandsvariablen oder ähnliches, die für die Vergleiche benötigt werden, und über den folgenden Spalten die abhängigen Ereignisse, die generiert werden, falls die Verknüpfungen innerhalb der entsprechenden Spalte den Wert TRUE ergeben. Die Operatoren, die für die Vergleiche zur Anwendung kommen, sind mit denen von Liste 11 identisch. Dadurch, daß die Liste der Verknüpfungen in zwei alternativen Formen angegeben wird, wird zum einen ein leichteres Protokollverständnis ermöglicht, und zum anderen eine der Implementation näher stehende Form angeboten.

Da diese Verknüpfungslisten bei komplexen Protokollen große Ausmaße annehmen können, werden in der Liste 13 nochmals alle abhängigen Ereignisse aufgeführt. Dies erfolgt ebenfalls automatisch durch das entwickelte Programmpaket zur

Protokollspezifikation und dient der besseren Übersicht über die generierten abhängigen Ereignisse.

Die Form der Liste 13, Liste der abhängigen Ereignisse, ist in Bild 6.6-5 dargestellt.

Nr.	abhäng. Ereignis	Bemerkung
E_xx	Ereignisname	erklärender Text
.	..	...
.	..	...

Bild 6.6-5 Liste der abhängigen Ereignisse

6.7 Definition der Zustandsübergänge

Nachdem ein Ereignis auf die Protokolleinheit getroffen ist, und alle Aktionen von dem Protokoll durchgeführt wurden, wird als letzte Aktion ein Zustandswechsel vorgenommen. In welchen Folgezustand die Protokolleinheit wechselt, hängt allein von dem aktuellen Zustand und dem, aus einem unabhängigen Ereignis generierten, abhängigen Ereignis ab.

Bild 6.7-1 zeigt in graphischer Form ein Beispiel für einige Zustandsübergänge, bewirkt durch abhängige Ereignisse, in einem Zustandsdiagramm.

In der Liste 14 der Protokollspezifikation, die Liste der Zustandsübergänge, wird für jeden Zustandswechsel der

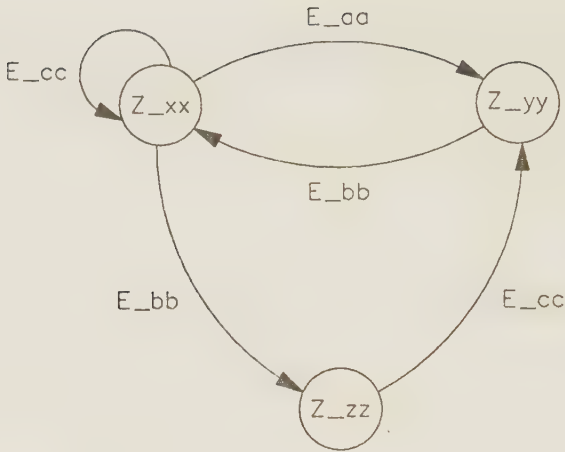


Bild 6.7-1 Zustandsdiagramm

aktuelle Zustand, das den Wechsel bewirkende Ereignis und der Folgezustand angegeben. Die Form der Liste 14 ist in Bild 6.7-2 dargestellt.

Aktueller Zust.	Ereignis	Folgezustand
Z_xx Name	E_yy Name	Z_zz Name
. ..	. ..	. ..
. ..	. ..	. ..

Bild 6.7-2 Liste der Zustandsübergänge

6.8 Aktionen der Protokolleinheit

Die Aktionen, die eine Protokolleinheit ausführen kann, werden in zwei Gruppen aufgeteilt. Es soll möglich sein, sofort nach Eintreten eines unabhängigen Ereignisses bestimmte Aktionen durchzuführen, unabhängig vom Zustand und den Zustandsvariablen des Protokolls. Diese Aktionen werden als zustandsunabhängige Aktionen bezeichnet. Sie werden nur in seltenen Fällen benötigt, z. B. wenn bei einem bestimmten Ereignis immer ein Timer gestoppt werden soll oder das Eintreten des Ereignisses bestätigt werden soll, bevor es verarbeitet wurde.

Nach Vorverarbeitung der unabhängigen Ereignisse treffen die generierten abhängigen Ereignisse auf das Protokoll in einem bestimmten Zustand und veranlassen die Protokolleinheit entsprechend diesem Zustand und dem abhängigen Ereignis Aktionen durchzuführen. Diese Aktionen werden als zustandsabhängige Aktionen bezeichnet.

6.8.1 Zustandsunabhängige Aktionen

Die zustandsunabhängigen Aktionen werden alleine von dem eingetretenen Ereignis bestimmt. Sie werden ausgelöst, bevor eine Vorverarbeitung der unabhängigen Ereignisse stattfindet. Dieser Zusammenhang ist in Bild 6.8.1-1 dargestellt.

Bei den zustandsunabhängigen Aktionen ist es möglich

- Schnittstellenaktionen auszulösen
- Timer zu starten oder zu stoppen.

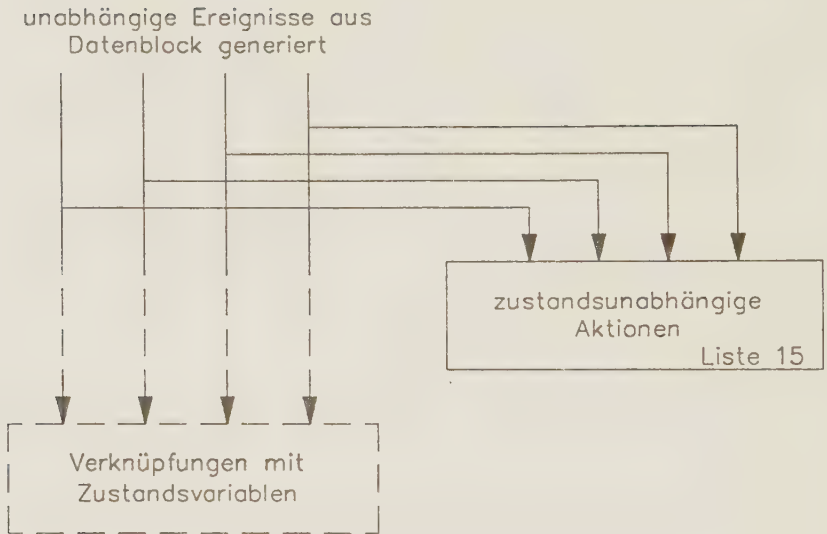


Bild 6.8.1-1 Auslösen von zustandsunabhängigen Aktionen

sowie folgende Daten zu modifizieren:

- Variablen
- Feldelemente oder Felder
- Common-Feldelemente oder Common-Felder
- Datenblockelemente.

Die möglichen Aktionen sind in Bild 6.8.1-2 aufgezeigt. Die zustandsunabhängigen Aktionen der Protokolleinheit werden in der Liste 15 definiert. Dabei stehen in der ersten Spalte dieser Liste die zu modifizierenden Datenelemente und

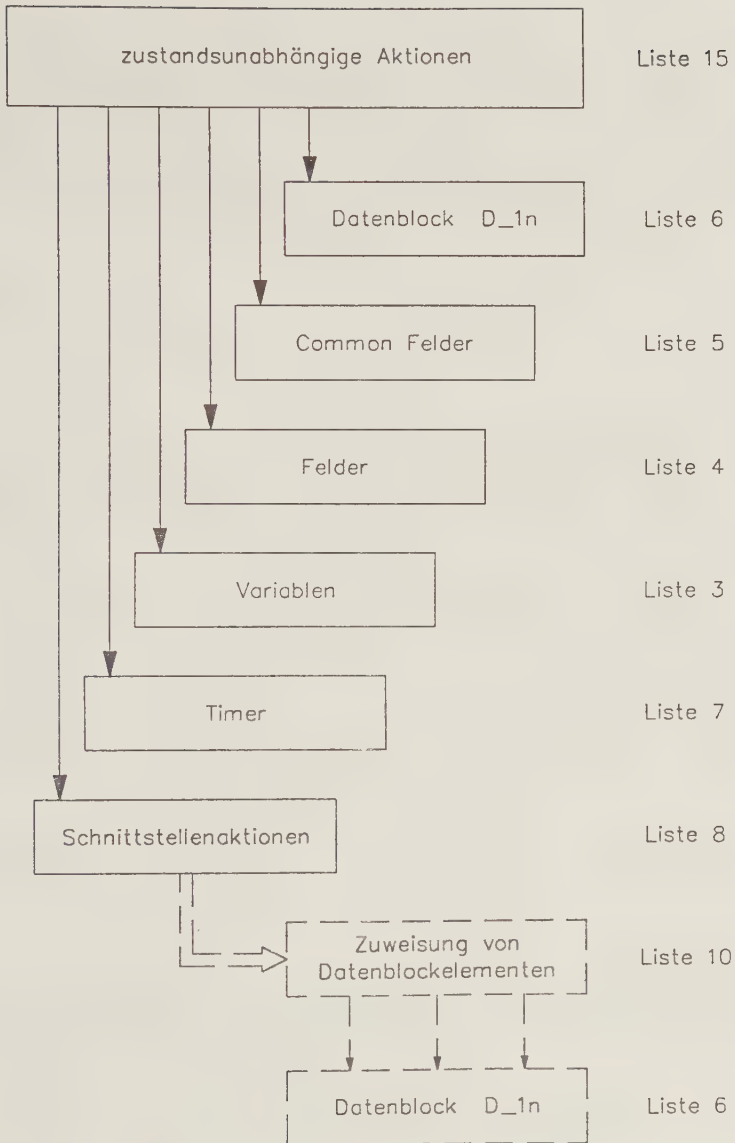


Bild 6.8.1-2 Zustandsunabhängige Aktionen

in den folgenden Spalte die Werte, die diesen Datenelementen zugewiesen werden sollen, bzw. die Operationen, die auf diese Daten anzuwenden sind. Über den jeweiligen Spalten wird das unabhängige Ereignis angegeben, bei dem die in der Spalte stehenden Aktionen auszulösen sind. Die Struktur der Liste 15, Liste der zustandsunabhängigen Aktionen, ist in Bild 6.8.1-3 dargestellt.

	Xn_xx	Xn_yy
Xn_yy	*	
V_xx	:= 15	-> 5
F_xx(F_yy(V_zz)	=* V_xx,K_yy	
Xn_zz		*
.	.	.
.	.	.

Bild 6.8.1-3 Liste der zustandsunabhängigen Aktionen

Die Operatoren, die bei den einzelnen Modifikationen der Daten zugelassen sind, werden im folgenden kurz beschrieben.

- * Die in der linken Spalte stehende Schnittstellenaktion wird ausgelöst

Das Auslösen einer Schnittstellenaktion ist immer die letzte aller Aktionen, die für das entsprechende unabhängige Ereignis definiert sind. Sind mehrere Schnittstellenaktionen

angegeben, so werden sie in der Reihenfolge ihres Auftretens innerhalb der Spalte ausgelöst, es sei denn , es wird durch den Operator

*n mit n eine Durchnumerierung der Schnittstellenaktionen angegeben.

Der Operator

:= Wertzuweisung

dient der direkten Wertzuweisung, wobei auch hier, wie schon in Liste 11 bei der Zuweisung von Datenblockelementen, ganze Felder oder Teile von Feldern zugewiesen werden dürfen. Dabei werden jeweils die Anfangselemente spezifiziert und angegeben, wie viele Elemente kopiert werden sollen.

Wird einem Timer (inneres Ereignis I_xx) ein von Null verschiedener Wert zugewiesen, so bedeutet dies, daß der Timer mit diesem Wert gestartet wird. Wird dem Timer der Wert Null zugewiesen, so wird der Timer gestoppt und läuft dadurch nicht mehr ab und löst somit kein unabhängiges Ereignis mehr aus. Die restlichen zugelassenen Operatoren

+	Addition
-	Subtraktion
**	Multiplikation
//	Division
/	Negation
&	bitweises logisches AND
/&	bitweises logisches NAND
\$	bitweises logisches OR
/\$	bitweises logisches NOR
<-	Rotation links
->	Rotation rechts
=&	Zuweisung der Adresse eines Feldes
=*	Zuweisung eines Feldes über Adresse

sind genauso definiert, wie im Abschnitt 6.5.2 bei der Zuweisung von Datenblockelementen in Liste 10 beschrieben. Im Gegensatz zu Liste 10 dürfen hier natürlich nicht nur Datenblockelemente, sondern alle in Bild 6.8.1-2 aufgeführten Elemente modifiziert werden.

6.8.2 Zustandsabhängige Aktionen

Nach Eintreten eines unabhängigen Ereignisses und dem Auslösen von Aktionen entsprechend diesem Ereignis werden, wie

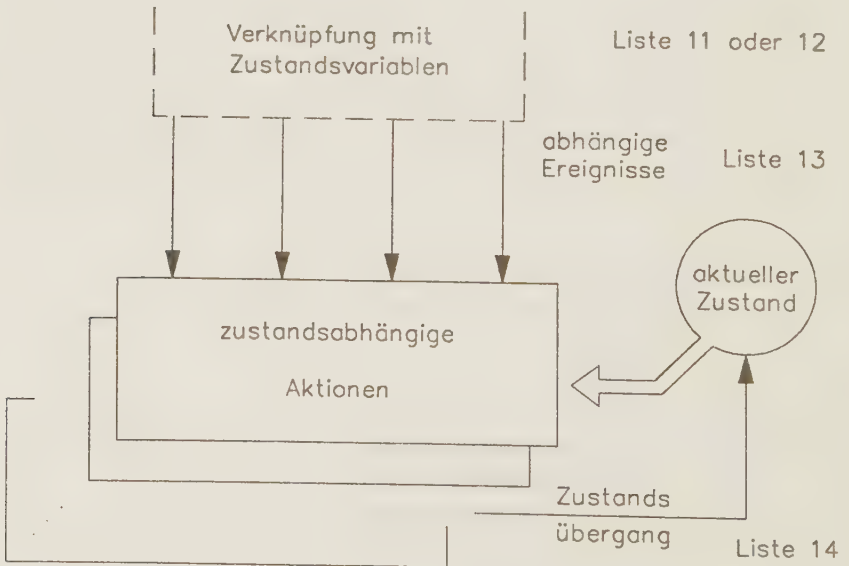


Bild 6.8.2-1 Auslösen von zustandsabhängigen Aktionen

in Abschnitt 6.6 beschrieben, abhängige Ereignisse generiert. Diese abhängigen Ereignisse treffen nun auf das Protokoll in einem bestimmten Zustand und lösen, abhängig von dem aktuellen Zustand, Aktionen aus, die als zustandsabhängige Aktionen bezeichnet werden. Bild 6.8.2-1 zeigt diese Vorgehensweise.

Als letzte Aktion wird der Zustandswechsel entsprechend Liste 14 in den Folgezustand vorgenommen.

Zu den zustandsabhängigen Aktionen der Protokolleinheit zählen:

- das Auslösen von Schnittstellenaktionen
- das Starten oder Stoppen von Timern
- das Modifizieren von protokollinternen Daten wie
 - Variablen
 - Felder
 - Common-Felder und
 - Datenblöcke.

Außerdem ist es möglich

- Funktionen aufzurufen
- eine Unterzustandsmaschine zu kreieren und zu löschen und
- ein unabhängiges Ereignis von einem SAP an eine Unterzustandsmaschine weiterzuleiten.

Die zustandsabhängigen Aktionen sind in Bild 6.8.2-2 dargestellt.

Auf die Benutzung und die Definition von Funktionen wird im Abschnitt 6.9 eingegangen. Die Definition von Unterzustandsmaschinen, deren Behandlung und das Weiterleiten von Ereignissen an die Unterzustandsmaschinen, sowie die Kommunikation zwischen Unterzustandsmaschinen und übergeordneter

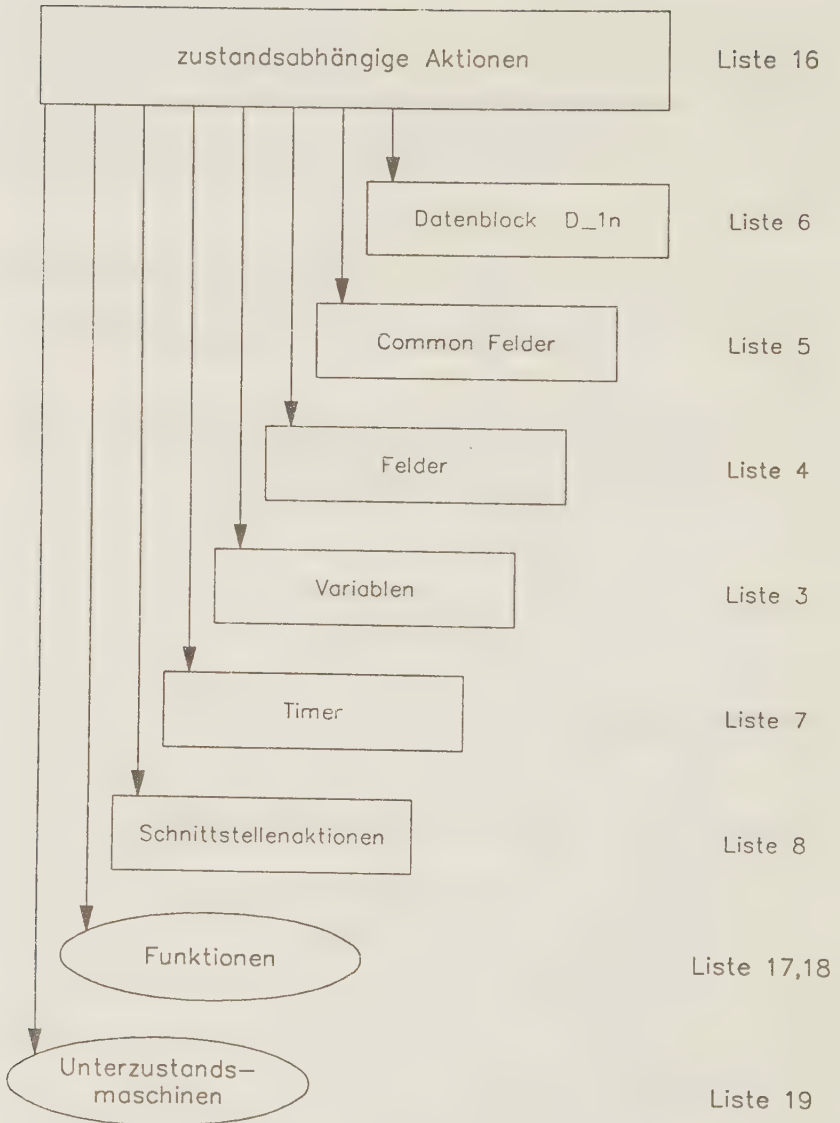


Bild 6.8.2-2 Zustandsabhängige Aktionen

Zustandsmaschine wird in Abschnitt 6.10 dargestellt und soll daher an dieser Stelle nicht näher erläutert werden.

Die Form der Liste 16, Liste der zustandsabhängigen Aktionen, ist in Bild 6.8.2-3 dargestellt.

Z_XX Offline

	E_xx	E_yy
Xn_yy	*	
V_xx	:= 15	-> 5
F_xx(F_yy(V_zz)	:=* V_xx,K_yy	
Xn_zz		*
.	.	.
.	.	.

Bild 6.8.2-3 Liste der zustandsabhängigen Aktionen

Über jeder Liste steht der Zustand, für den die Aktionen, die unter den abhängigen Ereignissen angegeben sind, gelten. Für jeden Zustand wird eine neue Liste angelegt.

Damit ist auch eindeutig spezifizierbar, daß das gleiche abhängige Ereignis in unterschiedlichen Zuständen unterschiedliche Aktionen auslösen kann.

Die in dieser Liste verwendeten Operatoren sind identisch mit denen, wie sie in der Liste 14 bei den zustandsunabhängigen Aktionen verwendet werden.

Bei dem Zuweisungsoperator (:=) darf nun außerdem eine Funktion P_xx (spezifiziert in Liste 17) angegeben werden.

Zusätzlich zu den Operatoren aus Liste 14 können in dieser Liste noch drei weitere Operatoren benutzt werden, die sich auf die Verwendung von Unterzustandsmaschinen beziehen und wie folgt spezifiziert sind:

- *+ Die angegebene Unterzustandsmaschine wird kreiert
- *- Die angegebene Unterzustandsmaschine wird gelöscht
- >> Der empfangene Datenblock vom SAP n, und damit auch das unabhängige Ereignis, werden an die angegebene Unterzustandsmaschine weitergegeben.

Eine nähere Erläuterung zu diesen Operatoren erfolgt ebenfalls in Kapitel 6.10

6.9 Definition von Funktionen

Für umfangreichere zustandsabhängige Aktionen oder für Aktionsfolgen, die häufiger benötigt werden, besteht die Möglichkeit Funktionen zu definieren. Funktionen werden mit dem Buchstaben P gekennzeichnet und mit einer zweistelligen Hexadezimalzahl durchnumeriert. Der Wertebereich des von der Funktion zurückgelieferten Wertes wird durch den Wertebereich der Variablen bestimmt, die hinter der Funktion angegeben wird. Die Definition der Funktion hat somit folgendes Aussehen:

$P_{xx}(V_{yy})$

Die Aktionen, die eine Funktion auslösen kann, sind identisch mit den zustandsabhängigen Aktionen. Auch die definierten Operatoren stimmen mit denen überein, wie sie bei den zustandsabhängigen Aktionen verwendet werden. Die hier definierten Funktionen sind nicht im Sinne einer

Funktion, wie sie in der Softwaretechnik definiert ist, zu verstehen, sondern mehr als die Zusammenfassung von einzelnen Aktionen. Funktionen können auf alle definierten Daten innerhalb der Zustandsmaschine zugreifen und diese modifizieren.

Die Funktionen werden in der Liste 17, Liste der Aktionen der Funktionen, definiert, deren Form in Bild 6.9-1 dargestellt ist.

P_xx(V_yy) Funktionsname		erklärender Text
Aktion 1		Aktion 2
Xn_yy	*	
V_xx	:= 15	-> 5
F_xx(F_yy(V_zz)	=* V_xx,K_yy	
Xn_zz		*
.	.	.
.	.	.

Bild 6.9-1 Liste der Aktionen der Funktionen

Der Aufbau dieser Liste ist ähnlich wie bei den zustandsabhängigen Aktionen, nur daß hier nacheinander auszuführende Aktionsfolgen angegeben werden können. Dies ist nützlich, wenn von einer Funktion mehrere Schnittstellenaktionen ausgelöst werden sollen, die teilweise gleiche Variablen oder den gleichen Datenblock benutzen.

Auch hier gilt wieder, daß wenn die Anzahl der Spalten nicht ausreicht, eine oder mehrere Listen mit den Folgespalten

angehängt werden.

Zur besseren Übersicht über die verwendeten Funktionen wird durch das Programmpaket zur Protokollspezifikation eine Liste der Funktionen generiert, in der jede definierte Funktion aufgeführt ist. Die Form dieser Liste ist in Bild 6.9-2 dargestellt.

Nr.	Name der Funktion	Bemerkungen
P_xx(V_yy)	Funktionsname	erklärender Text
.	..	...
.	..	...

Bild 6.9-2 Liste der Funktionen

Die Eintragungen in dieser Liste entsprechen genau den Funktionsdefinitionen aus Liste 17.

6.10 Definition der Unterzustandsmaschinen

Um die Möglichkeit zu erhalten, ein Protokoll hierarchisch zu entwerfen, wurden Unterzustandsmaschinen eingeführt. Eine Unterzustandsmaschine ist eine komplette Zustandsmaschine, die innerhalb eines Zustands einer übergeordneten Zustandsmaschine existiert. Dieser Zusammenhang ist in Bild 6.10-1 dargestellt.

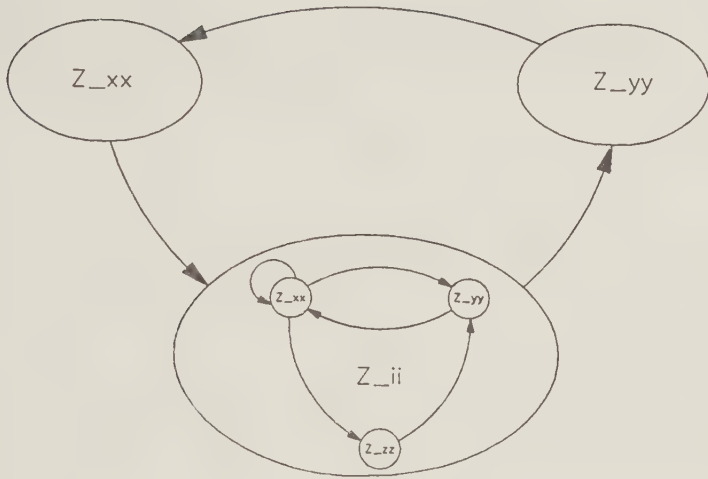


Bild 6.10-1 Unterzustandsmaschine als Teil eines übergeordneten Zustands

Unterzustandsmaschinen sollten dort eingeführt werden, wo in sich abgeschlossene Teilabläufe innerhalb einer Protokolleinheit in einem Zustand realisiert werden müssen. Eine Unterzustandsmaschine kann in dem übergeordneten Zustand auch mehrfach benutzt werden. Dies wird immer dann benötigt, wenn mehrere Teilabläufe innerhalb einer Protokolleinheit parallel bearbeitet werden müssen. Als Beispiel diene hierzu eine Protokolleinheit, die über eine physikalische Verbindung mehrere logische Verbindungen abwickelt. Wenn in einem Zustand des Protokolls logische Verbindungen aufgebaut werden, so wird das Protokoll für diese logische Verbindung am

sinnvollsten in Form einer Unterzustandsmaschine spezifiziert. Da der Protokollablauf für jede logische Verbindung gleich ist, muß in der übergeordneten Zustandsmaschine für jede Verbindung nur jeweils eine Unterzustandsmaschine für diese Verbindung kreiert werden, und die Ereignisse für die entsprechenden Verbindungen an die jeweilige Unterzustandsmaschine weitergeleitet werden.

Die Unterzustandsmaschine muß also in der Lage sein, ein komplettes Protokoll abzuwickeln, von dem Eintreffen von unabhängigen Ereignissen bis zum Auslösen von Schnittstellenaktionen. Eine Unterzustandsmaschinendefinition besteht daher aus den gleichen 19 Listen wie auch die Definition der übergeordneten Zustandsmaschine.

In der übergeordneten Zustandsmaschine wird die Unterzustandsmaschine nur über einen Namen definiert, der dann eine eigenständige Protokollspezifikation angibt. Bei dem Programmpaket zur Protokollspezifikation entspricht dieser Name auch gleichzeitig den Dateinamen, unter denen die einzelnen Listen für die Unterzustandsmaschine zu finden sind.

Unterzustandsmaschinen werden mit dem Buchstaben U gekennzeichnet. Es können 15 verschiedene Unterzustandsmaschinentypen definiert werden. Die Anzahl der Unterzustandsmaschinen, die pro Typ kreiert werden können, wird in Klammern hinter der Definition als Zahl oder Konstante angegeben. Die komplette Bezeichnung lautet somit

$$U_0n(i)$$

mit $n = 1..F$ und $i = \text{Zahl oder Konstante}$

Die Form der Liste 19, Liste der Unterzustandsmaschinen, ist in Bild 6.10-2 dargestellt.

Nr.	Dateiname	Bemerkungen
U_On(K_xx)	Name	erklärender Text
U_Om(i)	Name	erklärender Text
..	..	...
..	..	...

Bild 6.10-2 Liste der Unterzustandsmaschinen

In einer Unterzustandsmaschine können wiederum neue Unterzustandsmaschinen definiert werden, so daß eine schrittweise Verfeinerung der gesamten Protokollspezifikation gegeben ist.

Ein Beispiel dazu ist in /FRE86/ zu finden, wo eine Protokolleinheit für einen File-Transfer in einem Netz 16 verschiedene Verbindungen gleichzeitig bearbeiten kann. Eine Verbindung wird dabei als Unterzustandsmaschine spezifiziert, die dann 16 mal kreiert werden kann. Innerhalb jeder dieser Verbindungen ist es möglich, 16 verschiedene Dateien zu bearbeiten, deren Zustand wiederum in einer Unterzustandsmaschine festgehalten wird.

Die Datenblöcke, die die Protokolleinheit über die SAPs erreichen, und die Datenblöcke, die von dem Protokoll versendet werden, sind in der Hauptzustandsmaschine und in allen Unterzustandsmaschinen gleich, da sie die Verbindung zu den Nachbarebenen darstellen und ein festes Format haben müssen. Sie werden deshalb bei der Definition einer Unterzustandsmaschine automatisch von der übergeordneten Zustandsmaschinendefinition übernommen und von dort in die Listen der Unterzustandsmaschine kopiert. Das gleiche gilt für die Common-Felder, die ebenfalls in allen Zustandsmaschinen gleich sind, und eine Möglichkeit bieten, gemeinsame Daten-

bereiche anzulegen, die von allen Zustandsmaschinen gelesen und modifiziert werden können.

6.10.1 Weiterleiten von Ereignissen an Unterzustandsmaschinen

Die unabhängigen Ereignisse, die das Protokoll in Form eines Datenblocks erreichen, müssen, wenn festgestellt wird, daß sie für eine Unterzustandsmaschine bestimmt sind, an diese weitergeleitet werden. Dies erfolgt nur bei den zustandsabhängigen Aktionen oder bei den Aktionen der Funktionen, da eine Unterzustandsmaschine ja nur innerhalb eines übergeordneten Zustands existent ist. Zum Weiterleiten des Ereignisses genügt es, den aktuell empfangenen Datenblock an die Unterzustandsmaschine zu übergeben, da diese aus dem Datenblock selbst ihre unabhängigen Ereignisse entnimmt, die nicht mit denen der übergeordneten Zustandsmaschine übereinstimmen müssen. Abgelaufene Timer werden direkt an die Unterzustandsmaschine weitergeleitet. Die Anweisung für das Weiterleiten eines Ereignisses bei den Aktionen hat folgendes Aussehen:

```
U_0n(xx) >> D_0m
```

Dies bedeutet, daß der am SAP m empfangene Datenblock an die Unterzustandsmaschine xx vom Typ n weitergeleitet wird. Diese Form des Weiterleitens eines Ereignisses an eine Unterzustandsmaschine wird nur dann angewendet, wenn die Unterzustandsmaschine bereits existiert. Ist dies nicht der Fall, so muß diese zunächst einmal kreiert werden, wobei ebenfalls der Datenblock weitergeleitet wird, da dort Informationen zur Initialisierung der Unterzustandsmaschine enthalten sein können. Das Kreieren einer Unterzustandsmaschine

erfolgt mit der Anweisung:

```
U_On(xx)  *+  D_0m
```

Äquivalent zum Kreieren einer Zustandsmaschine erfolgt das Löschen mit der Anweisung

```
U_On(xx)  *-  D_0m
```

Auch hierbei wird das Ereignis weitergeleitet, da vor dem tatsächlichen Löschen der Unterzustandsmaschine noch Aktionen von dieser ausgelöst werden können. Als Beispiel dazu diene das oben genannte Protokoll für File-Transfer, wo vor dem Löschen einer Verbindung noch alle offenen Dateien dieser Verbindung geschlossen werden sollen.

Soll der ganze Mechanismus flexibel gestaltet werden, indem der Unterzustandsmaschine die Möglichkeit gegeben wird, anhand von Aktionen und Ereignissen selbst zu entscheiden, ob sie gelöscht wird oder nicht, dann taucht ein Problem auf, wie dies dem übergeordneten Zustand mitgeteilt werden kann. Es muß also ein Kommunikationsmechanismus zwischen Unterzustandsmaschine und übergeordneter Zustandsmaschine geschaffen werden, woran die übergeordnete Zustandsmaschine erkennt, ob eine Unterzustandsmaschine existiert oder gelöscht ist. Für diesen Kommunikationsmechanismus gibt es mehrere denkbare Möglichkeiten.

6.10.2 Kommunikation zwischen Unterzustandsmaschine und übergeordneter Zustandsmaschine

Eine Möglichkeit, eine Kommunikation zwischen den Zustandsmaschinen zu realisieren, wären die in Liste 5 definierten Common-Felder. Dies würde jedoch bedeuten, da diese Common-

Felder von allen Zustandsmaschinen gemeinsam benutzt werden können, daß ein sehr hoher Aufwand betrieben werden muß, um bestimmte Feldelemente bestimmten Unterzustandsmaschinen zuzuordnen, zumal wenn die Unterzustandsmaschinen selbst wieder Unterzustandsmaschinen definieren. Ein weiterer Nachteil bestünde in der Möglichkeit, daß eine Unterzustandsmaschine nicht nur sich selbst, sondern auch beliebige andere, auch ihr übergeordnete, löschen könnte. Dieser Mechanismus sollte von vornherein durch die Struktur des Protokollspezifikationsverfahrens vermieden werden.

Die Vorgehensweise, alle Unterzustandsmaschinen immer existent zu halten, und alle Ereignisse immer an diese weiterzuleiten, so daß die Unterzustandsmaschinen dafür verantwortlich sind, wie auf die Ereignisse zu reagieren ist, oder ob sie überhaupt verarbeitet werden können, wurde ausgeschlossen. Dies verhindert bei der späteren Realisierung der Protokolleinheit eine dynamische Vergabe von Ressourcen, und die Möglichkeit, entsprechend den jeweiligen Anforderungen den Speicherplatz für die Unterzustandsmaschinen zu verwalten und für nicht benutzte Unterzustandsmaschinen den Speicherplatz wieder freizugeben.

Eine andere Möglichkeit die Kommunikation zu realisieren wäre in der übergeordneten Zustandsmaschine für jeden Unterzustandsmaschinentyp eine Funktion zu reservieren, die von der Unterzustandsmaschine aufgerufen werden kann. Bild 6.10.2-1 zeigt diese Überlegung.

In der übergeordneten Zustandsmaschine müßte für jeden Unterzustandsmaschinentyp ein Feld definiert werden, in dem der Status der einzelnen Unterzustandsmaschinen festgehalten wird. Diese Feldelemente könnten dann von der Funktion modifiziert werden. Diese Vorgehensweise funktioniert auch bei beliebig tief verschachtelten Unterzustandsmaschinen, hat aber nach wie vor den Nachteil, daß eine Unterzustandsmaschine eine andere löschen kann, oder sogar beliebige

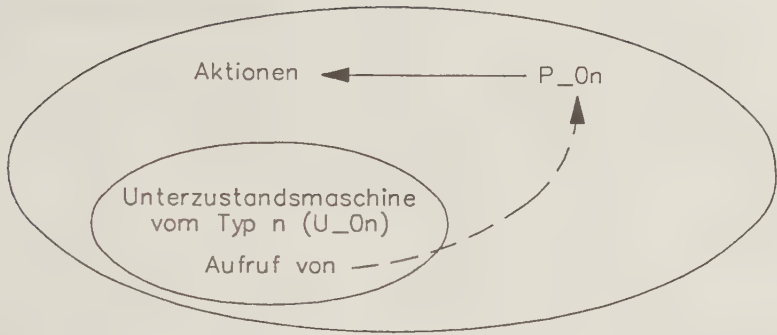
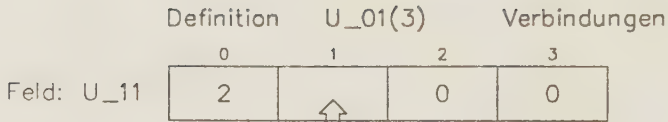


Bild 6.10.2-1 Aufruf von Funktionen durch
Unterzustandsmaschinen

Aktionen in dem übergeordneten Zustand durchführen kann. Dies könnte vermieden werden, wenn dafür gesorgt wird, daß die Unterzustandsmaschine nur auf das ihr entsprechende Feldelement zugreifen kann.

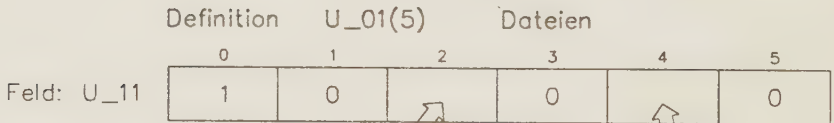
Aus diesem Grund wurde folgender Mechanismus eingeführt: In der übergeordneten Zustandsmaschine wird für jeden Unterzustandsmaschinentyp ein Feld definiert, mit genau so vielen Elementen wie es Unterzustandsmaschinen dieses Typs gibt. Jedes Feldelement entspricht einer Unterzustandsmaschine und enthält TRUE oder FALSE, je nachdem, ob die Unterzustandsmaschine existiert oder nicht. In der Unterzustandsmaschine ist eine Variable definiert, die identisch ist mit ihrem

Hauptzustandsmaschine



U_00

Untenzustandsmaschine
Verbindungen



U_00

Untenzustandsmaschine
Dateien

U_00

Untenzustandsmaschine
Dateien

Bild 6.10.2-2 Kommunikation zwischen Zustandsmaschinen

korrespondierenden Feldelement in der übergeordneten Zustandsmaschine. Die Bezeichnung für das Feld lautet:

$U_{ln}(i).$

Es existiert immer automatisch parallel zur Definition der Unterzustandsmaschinen

$U_{0n}(i)$

und braucht in der Liste nicht angegeben zu werden. Die Bezeichnung der Variablen, mit der die Unterzustandsmaschine auf ihr Feldelement in der übergeordneten Zustandsmaschine zugreift, lautet:

U_{00}

Auch diese Variable ist automatisch in jeder Unterzustandsmaschine enthalten und braucht nicht definiert zu werden.

Zur Verwaltung der Unterzustandsmaschinen und einem leichteren Finden einer unbenutzten Unterzustandsmaschine enthält das Feldelement $U_{ln}(00)$ immer die Nummer der ersten unbenutzten Unterzustandsmaschinen dieses Typs oder eine Null, falls alle benutzt sind. In Bild 6.10.2-2 ist dieser Kommunikationsmechanismus nochmals dargestellt.

Da per Definition alle Werte außer Null TRUE entsprechen, könnte bei der Realisierung in die Feldelemente, die angeben, ob eine Unterzustandsmaschine existiert, ein Zeiger auf den Datenbereich der Unterzustandsmaschine eingetragen werden. Dies vereinfacht das Kreieren und Löschen der Unterzustandsmaschine durch die übergeordnete Zustandsmaschine.

7 Programmpaket zum Entwurf der Protokollspezifikation

Wie im vorangegangenen Abschnitt gezeigt, besteht eine Protokollspezifikation aus bis zu 19 verschiedenen Listen. Diese Listen sollen in einen Rechner eingegeben werden, damit sie später von dem Rechner weiterverarbeitet werden können. Zu diesem Zweck wurde ein Programmpaket geschaffen, das das Eintragen der Listen im interaktiven Dialog mit dem Benutzer vornimmt. Von diesem Programm wird automatisch eine Protokolldokumentation erstellt, und es werden erste Überprüfungen auf die Korrektheit der Protokollspezifikation vorgenommen. Eine Menueoberfläche hilft, den Benutzer durch die einzelnen Phasen des Protokollentwurfs und der Spezifikation zu führen und bietet zu jedem Zeitpunkt Hilfestellungen an.

7.1 Benutzeroberfläche des Programmpakets

Die Benutzeroberfläche soll dem Anwender helfen, sich in das Protokollentwurfsverfahren einzuarbeiten und Hilfestellungen anzubieten. Es wurde daher eine menuegesteuerte Benutzeroberfläche gewählt, die den Anwender schrittweise durch die einzelnen Punkte des Protokollentwurfsverfahrens führt, dem erfahrenen Anwender jedoch gleichzeitig die Möglichkeit bietet, die einzelnen Punkte direkt anzuwählen.

Nach dem Starten des Programms wird der Benutzer aufgefordert, den Namen der Protokollspezifikation, die er bearbeiten möchte, anzugeben (vgl. Bild 7.1-1). Dieser Name ist gleichzeitig der Name der Dateien, unter denen die Spezifikation abgelegt wird, wobei durch das Programm noch selbsttätig Erweiterungen an diesen Namen, zur Unterscheidung der einzelnen Listen, angehängt werden. Es besteht dann die Möglichkeit diese Spezifikation zu bearbeiten, neu zu erstellen oder eine vorhandene Spezifikation zu drucken. Der

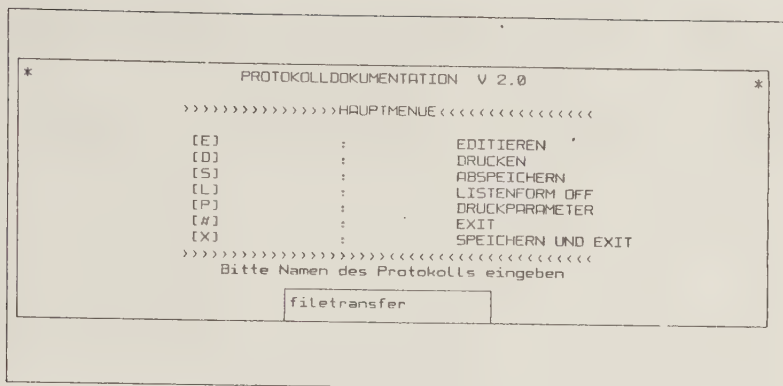


Bild 7.1-1 Startmenue

```

[#: HAUPTMENUE
(E) [1]: LISTE DER ZUSTAENDE
(/) [2]: LISTE DER KONSTANTEN
(/) [3]: LISTE DER VARIABLEN
(/) [4]: LISTE DER FELDER
(/) [5]: LISTE DER COMMON - FELDER
(/) [6]: LISTE DER DATENBLOCKE
(/) [7]: LISTE DER UNABHAENIGEN TRIGGEREREIGNISSE
(/) [8]: LISTE DER SCHNITTSTELLENAKTIONEN
(/) [9]: VERKNUEPFUNG VON DATENBLOCKEL. ZU UNAB. TRIGGEREREIGNISSEN
(/) [A]: ZUEWISUNG VON DATENBLOCKELEMENTEN BEI SCHNITTSTELLENAKTION
(/) [B]: LISTE DER VERKNUEPFUNGEN NACH ANH. EREIGNISSEN AUSGERICHTET
(/) [C]: LISTE DER VERKNUEPFUNGEN NACH UNABH. EREIGNISSEN AUSGER.
(/) [D]: LISTE DER ABHAENIGEN TRIGGEREREIGNISSE
(/) [E]: LISTE DER ZUSTANDSUEBERGANGE
(/) [F]: LISTE DER ZUSTANDSUNABHAENIGEN AKTIONEN
(/) [G]: LISTE DER ZUSTANDSABHAENIGEN AKTIONEN
(/) [H]: LISTE DER AKTIONEN DER FUNKTIONEN
(/) [I]: LISTE DER FUNKTIONEN
(/) [J]: DEFINITION DER UNTERZUSTANDSMASCHINEN

```

Anwahl durch Cursortasten oder Eingabe des Kennbuchstabens und (CR)

Bild 7.1-2 Bildschirm für Listenauswahl

Ausdruck der Spezifikation kann auf unterschiedlichen Druckern erfolgen, die vom Anwender installiert werden können.

Bei der Bearbeitung der Spezifikation wird dem Benutzer eine Auswahl aller zu bearbeitenden Listen vorgelegt, wobei in der ersten Spalte gekennzeichnet wird, ob diese Liste während des aktuellen Programmlaufs schon bearbeitet wurde (Bild 7.1-2). Da die Struktur einzelner Listen sich sehr ähnelt, soll die Vorgehensweise des Eintragens in diese Listen an zwei Beispielen exemplarisch dargestellt werden. Eine genaue Beschreibung und ein Benutzerhandbuch zu diesem Programmpaket sind in /TEG88/ beschrieben.

Als erstes Beispiel diene die Liste der Zustände, die, wie in Abschnitt 6.1 beschrieben, pro Zustand nur aus drei Einträgen besteht. Dies sind die Zustandsnummer, der Name des Zustands und eine Bemerkung. Auf dem Bildschirm werden für diese Einträge entsprechende "Fenster" geöffnet, in denen der Text eingetragen wird (Bild 7.1-3). Schon vorhandene

The image shows a screen layout for a simple list. It contains three main input fields: 'ZUSTAND' with a small box containing 'Z_', 'NAME' with a larger empty box, and 'BEMERKUNG' with a long empty box. To the right of these fields is a menu box with the title 'AUSWAHL' and four options: 'EDIT', 'VIEW', and 'ENDE'. The menu is separated from the other fields by dashed lines.

ZUSTAND	NAME	BEMERKUNG
Z_		

AUSWAHL

EDIT

VIEW

ENDE

Bild 7.1-3 Bildschirmdarstellung für einfache Listen

Einträge in dieser Liste können wieder verändert werden. Es besteht die Möglichkeit, sich alle Einträge der Liste auf dem Bildschirm anzeigen zu lassen und auch dabei Veränderungen an den Daten vorzunehmen (Bild 7.1-4).

ZUSTAND	NAME
Z_00	Offline

BEMERKUNG

uninitialisiertes Protokoll

AUSWAHL

AUSWAHL

NEXT

DELETE

ENDF

Bild 7.1-4 Bildschirmdarstellung beim Durchblättern der Liste der Zustände

Diese Art des Eintragens gilt für alle Listen, in denen sequentiell Einträge nacheinander vorgenommen werden. Beispiele dazu sind u.a. die Liste der Konstanten, Variablen oder Felder. Die Einträge in diesen Listen werden automatisch nach der Nummer des Elements, das in diesen Listen definiert wird, sortiert.

Eine etwas andere Vorgehensweise wurde bei den Listen gewählt, die zum einen aus mehreren Spalten bestehen, was z. B. auf die Liste der Verknüpfungen zutrifft, oder wo es auf die Reihenfolge der einzelnen Anweisungen, wie bei den Aktionen der Protokolleinheit, ankommt.

LISTE DER AKTIONEN DER FUNKTIONEN

FUNKT. VARIABLE	NAME DER FUNKTION	OPTIONALE BEMERKUNG
P_00 (V_14)	reset_timer	alle Timer stoppen

1

EINTRAG
OP
WERT

I_00

:=

0

AKTION 1

FUNKTION
AKTION
EINTRAG
VIEW
ENDE

Bild 7.1-5 Bildschirmdarstellung bei komplexeren Listen

LISTE DER AKTIONEN DER FUNKTIONEN

FUNKT. VARIABLE	NAME DER FUNKTION	OPTIONALE BEMERKUNG
P_00 (V_14)	reset_timer	alle Timer stoppen

2

EINTRAG
OP
WERT

I_01

:=

0

AKTION 1

FUNKTION
AK
EI
V
E

AUSWAHL
FU
AK
EI
EN

NEXT
INSERT
DELETE
MOVE
EDIT
ENDE

Bild 7.1-6 Bildschirmdarstellung beim Durchblättern und Modifizieren komplexerer Listen

Ein Beispiel für diese Art der Listen sei die Liste der Aktionen der Funktionen, die einerseits aus mehreren Spalten für verschiedene Aktionsketten besteht, wobei gleichzeitig aber auch die Reihenfolge der Aktionen innerhalb einer Spalte eingehalten werden muß. Bild 7.1-5 zeigt den Bildschirmaufbau zum Bearbeiten dieser Liste.

Hier können die Funktion, die Aktionsnummer und der Eintrag jeweils getrennt voneinander angewählt werden. Alle Eingaben erfolgen in entsprechenden Fenstern auf dem Bildschirm. Bestehende Einträge können nachträglich geändert werden, und auch hier kann der Benutzer sich die komplette Liste auf dem Bildschirm anzeigen lassen. Es kann dann durch die einzelnen Einträge "geblättert" werden (Bild 7.1-6), wobei weitergehende Bearbeitungsmöglichkeiten wie das Einfügen, Löschen oder Verändern von Einträgen und das Verschieben von Einträgen und Aktionsketten an andere Positionen zur Verfügung gestellt werden.

Während des Eintragens der Listen werden schon erste Überprüfungen auf sinnvolle Eingaben durch das Programm durchgeführt. Dies beinhaltet beispielsweise die Überprüfung auf legale Operatoren bei Zuweisungen oder Vergleichen, oder auch die Adressierung komplexer Datenstrukturen wie Felder und Datenblöcke.

7.2 Innere Struktur des Programmpakets

Die Struktur des Programms soll in dieser Arbeit nur kurz aufgezeigt werden. Eine vollständige Beschreibung mit Hierarchiediagrammen und Struktogrammen befindet sich in /TEG88/.

Der Entwurf des Programms erfolgte nach einer Top-Down Vorgehensweise. Das Programm ist modular gegliedert, mit eindeutig spezifizierten Schnittstellen der einzelnen Module.

Da die Größe der einzelnen Listen im voraus nicht abzuschätzen ist, und durch Vorgabe einer Maximallänge einerseits keine Einschränkung vorgenommen werden sollte, andererseits aber auch nicht Speicherplatz gebunden wird, der dann doch nicht durch Daten belegt ist, wurden sämtliche Daten zur Beschreibung der Listen in Form dynamischer Datenstrukturen aufgebaut. Dadurch ist gewährleistet, daß der benötigte Speicherplatz den jeweiligen Erfordernissen der aktuellen Protokollspezifikation angepaßt ist. Sämtliche Längen von Zeichenketten oder ähnlichen Strukturen wurden als Konstanten innerhalb des Programms vereinbart, so daß nachträgliche Änderungen schnell durchführbar sind. Damit solche Änderungen später durch das alleinige Ändern einer Konstanten durchgeführt werden können, muß diese Konstante natürlich bei allen Operationen, die sich auf die Maximallänge einer durch sie definierten Struktur beziehen, benutzt werden. Dies erfolgt bis zur Festlegung der Breite der Eingabefenster, in denen die Daten vom Benutzer eingegeben werden. Gleiches gilt für die Überprüfung auf Fehler während der Eingabe. Auch hier sind alle legalen Operatoren und alle Bereichsgrenzen als Konstanten vereinbart, so daß eine spätere Änderung oder Erweiterung an zentraler Stelle durch das Umdefinieren von Konstanten und eine Neuübersetzung des Programms erfolgen kann.

Die Übersetzung des Programmpakets erfolgt automatisch durch eine Kommandodatei, in der alle Abhängigkeiten der Module untereinander definiert sind, und wo anhand des Datums und der Uhrzeit der zum Programm gehörenden Dateien festgestellt werden kann, welche Module vor dem Binden des Programms neu übersetzt werden müssen.

7.3 Syntaxüberprüfung der Protokollspezifikation

Nach erfolgter Protokollspezifikation besteht die Möglichkeit eine erste Überprüfung auf syntaktische Richtigkeit des Protokolls durch das Programm durchführen zu lassen. Dabei handelt es sich nicht um eine Überprüfung der Korrektheit des logischen Protokollablaufs, sondern um eine Kontrolle der Struktur der Protokollspezifikation.

Die einzelnen Aspekte, die dabei überprüft werden, sollen im folgenden kurz aufgezeigt werden.

Zunächst wird überprüft, ob alle Zustände, die bei den Zustandsübergängen benutzt werden, auch in der Liste der Zustände definiert sind, und ob Zustände definiert sind, die jedoch bei den Zustandsübergängen nicht benutzt werden. Danach findet eine Überprüfung auf isolierte Zustände statt, das heißt Zustände, die zwar durch Zustandsübergänge erreicht werden, in denen aber keine Ereignisse mehr definiert sind, oder Zustände, die nie nach einem Zustandsübergang eingenommen werden.

Diese Kontrollen sagen noch nichts darüber aus, daß die Zustandsfolgen korrekt definiert sind, geben aber, falls bei einem der oben genannten Punkte eine Unkorrektheit festgestellt wird, einen Hinweis auf mögliche Fehler in der Protokollspezifikation.

In einem nächsten Schritt wird überprüft, ob alle Daten wie Konstanten, Variablen, Datenblöcke, Felder und Common-Felder, die in der Spezifikation benutzt werden, auch in den entsprechenden Listen definiert sind, und ob Datenstrukturen definiert sind, die im folgenden nicht benutzt werden. Eine unbenutzte Variable wäre zwar kein Fehler im Protokoll, deutet aber auf einen Fehler, möglicherweise einen Tippfehler beim Eingeben, hin.

Eine Überprüfung auf Wertebereichsüberschreitungen bei Zuweisungen läßt sich nicht so einfach durchführen, da die Variablenwerte erst zur Laufzeit des Protokolls feststehen. Bei der Zuweisung von Konstanten und Zahlen wird diese Über-

prüfung jedoch vorgenommen, und es wird kontrolliert, ob der Wertebereich einer Variablen kleiner ist als der Wertebereich der Variablen, die ihr zugewiesen wird. Die gleichen Überprüfungen werden auch auf Felder oder Feldelemente und die übrigen definierten Datenstrukturen angewandt.

Die Operatoren, die in den Listen für Vergleiche und Zuweisungen benutzt werden, werden, wie bereits erwähnt, schon während der Eingabe auf Korrektheit überprüft, und auch darauf, ob der angegebene Operator für die beiden Operanden eine legale Operation darstellt.

In einem letzten Schritt wird die Vollständigkeit der Ereignisse und Schnittstellenaktionen der Protokolleinheit überprüft. Dies bedeutet, daß alle definierten unabhängigen Ereignisse, die auf das Protokoll wirken, auch benutzt werden oder ob Ereignisse benutzt werden, die nicht definiert sind. Unter der Voraussetzung, daß die Liste der unabhängigen Ereignisse, die ja, wie im vorigen Kapitel erwähnt, von den Schnittstellenaktionen der Nachbarebene abhängt, korrekt ist, kann dann davon ausgegangen werden, daß die Protokolleinheit zumindest auf jedes auf sie wirkende Ereignis reagiert. Eine ähnliche Überprüfung wird anschließend auf die abhängigen Ereignisse, die aus den unabhängigen generiert werden, angewandt. Hierbei wird überprüft, ob alle Ereignisse, die generiert werden, in der Liste der Zustandsübergänge auch einen Zustandsübergang bewirken, wobei Ausgangszustand und Folgezustand durchaus identisch sein können.

Bei den Schnittstellenaktionen wird festgestellt, ob alle definierten Schnittstellenaktionen auch ausgelöst werden.

Die Überprüfung auf Korrektheit der unabhängigen Ereignisse und der Schnittstellenaktionen beinhaltet auch die Listen, die den Zusammenhang zwischen diesen Ereignissen und Aktionen und den Datenblöcken beschreiben.

7.4 Simulation der Protokollspezifikation

Sinn einer Protokollsimulation auf Basis der Protokollspezifikation ist, in einem sehr frühen Stadium der Protokollentwicklung, Fehler im Protokoll zu finden. Werden Fehler im Protokoll erst beim Testen nach erfolgter Implementation erkannt, so muß nicht nur die Protokollspezifikation korrigiert werden, sondern auch die Implementation der Protokolleinheit, was mit einem erheblichen Aufwand verbunden sein kann. Eine fehlerfreie Protokollspezifikation ist besonders wichtig bei internationalen Normen für Protokolle einer offenen Kommunikation, da diese Normen die Implementationsgrundlage für die verschiedensten Anwender darstellen.

In dieser Arbeit soll nun nicht auf die unterschiedlichen Test- und Simulationsstrategien für Protokolle eingegangen werden, vielmehr soll aufgezeigt werden, in welcher Weise eine Protokollspezifikation, die nach der beschriebenen Methode erstellt ist, für eine Simulation der Protokollspezifikation geeignet ist.

Dazu wurde ein Programm entwickelt, das eine Simulation der Spezifikation rechnergestützt ermöglicht. Die Arbeitsweise des Simulationsprogramms läßt sich folgendermaßen beschreiben.

Zunächst werden sämtliche Listen, die zur Protokollspezifikation gehören, inclusive der Unterzustandsmaschinen, eingelesen. Danach werden von dem Simulationsprogramm alle in diesen Listen definierten Datenstrukturen angelegt. Dies erfolgt, da der Umfang der Daten nicht vorhergesagt werden kann, in Form dynamischer Strukturen. Innerhalb des Simulationsprogramms existiert jetzt eine Abbildung der Protokollspezifikation in Form der eingelesenen Listen und der Daten, die von der Protokolleinheit benötigt werden. Für die Simulation des Protokolls stehen nun drei Möglichkeiten zur Verfügung.

Die erste Möglichkeit ist in Bild 7.4-1 dargestellt. Dabei wird der Bildschirm des Terminals in drei Bereiche (Fenster)

aufgeteilt. Das erste Fenster dient dem Auslösen von Ereignissen für die Protokolleinheit, d.h. es können Datenblöcke für einen bestimmten SAP aufgebaut werden. Dieser Datenblock wird von dem Simulationsprogramm übernommen, und es wird daraus, entsprechend der Liste 9 (Verknüpfung von Datenblockelementen zu unabhängigen Ereignissen) ein unabhängiges Ereignis generiert. Entsprechend dem Protokollablauf und dessen Beschreibung in den Listen werden nun sämtliche Abläufe simuliert, wobei den Datenstrukturen des Protokolls die entsprechenden Werte zugewiesen werden. Dies erfolgt so lange, bis alle Aktionen, die in den Listen definiert sind, abgearbeitet sind. Bei Schnittstellenaktionen, die in dem Versenden eines Datenblocks enden, wird der versendete Datenblock im Fenster 2 des Bildschirms dargestellt.

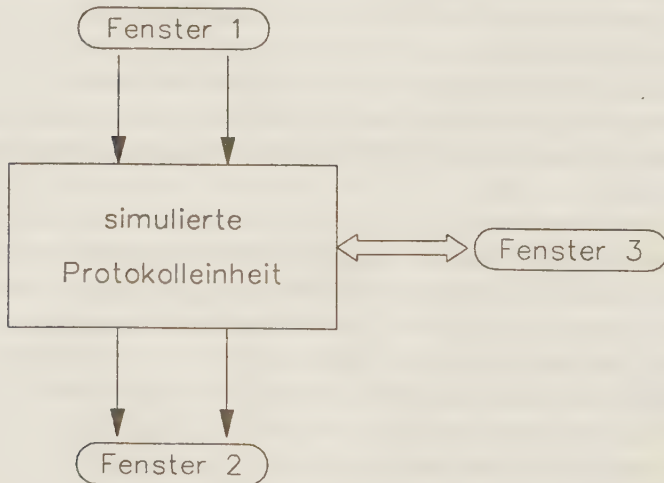


Bild 7.4-1 Erste Möglichkeit der Protokollsimulation

Der dritte Bildschirmbereich dient der Steuerung und der Überwachung des Simulationsprogramms, wobei die Möglichkeit besteht, den inneren Protokollablauf zu verfolgen und beliebige Daten des Protokolls zu überwachen.

Der Protokollablauf sollte an den Schnittstellen identisch sein mit dem einer späteren Implementation, mit dem Unterschied, daß das Protokoll nicht implementiert und codiert ist, sondern von dem Simulationsprogramm interpretiert wird. Die Ereignisse für das Protokoll müssen in dieser Stufe der Simulation noch für alle SAPs vom Benutzer über den Bildschirmbereich 1 eingegeben werden.

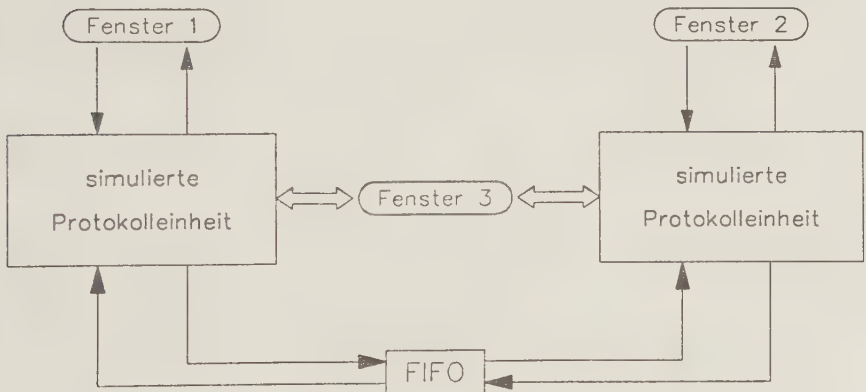


Bild 7.4-2 Simulation mit zwei Protokolleinheiten

In einer weiteren Stufe der Simulation, die in Bild 7.4-2 dargestellt ist, wird die Simulation auf zwei Protokolleinheiten, die miteinander kommunizieren, ausgedehnt. Die Listen, die für die Protokolle zu interpretieren sind, sind für

beide Kommunikationspartner identisch, lediglich die Datenstrukturen müssen doppelt, d.h. für jede Protokolleinheit getrennt, ausgelegt werden. Die Simulation könnte auch durch zwei unterschiedliche Prozesse, die untereinander Daten austauschen, realisiert werden, um ein quasi paralleles Abarbeiten der Protokolleinheiten zu erreichen. Das parallele Abarbeiten der Protokolle kann jedoch auch von dem Simulationsprogramm durchgeführt werden, mit dem Vorteil, daß die Listen, die den Protokollablauf beschreiben, nur einmal in den Speicher eingelesen werden müssen.

Die beiden simulierten Protokolleinheiten wickeln nun das Schichtenprotokoll tatsächlich untereinander ab, da alle Schnittstellenaktionen zu SAPs der tieferen Ebene über einen Puffer an die andere Protokolleinheit weitergeleitet werden. Die Ein- und Ausgaben über die SAPs zur höheren Ebene werden in den Fenstern 1 und 2 des Bildschirms dargestellt. Das Fenster 3 dient wiederum der Steuerung und Überwachung des Simulationsablaufs.

In einer letzten Stufe der Simulation wird nun noch für eine Protokolleinheit die höhere Ebene nachgebildet. Dies geschieht in Form einer Datei, in der für jede Aktion der Protokolleinheit ein Ereignis definiert ist, das nach dem Auslösen der Schnittstellenaktion auf die Protokolleinheit wirken soll. Dabei kann eine Schnittstellenaktion in dieser Datei mehrfach auftreten, um unterschiedliche Ereignisse auf das Protokoll wirken zu lassen, und es können auch Ereignisketten spezifiziert werden, wenn pro Aktion mehr als ein Ereignis auf das Protokoll wirken soll. Diese Vorgehensweise ist in Bild 7.4-3 nochmals dargestellt.

Durch die Simulation der Protokollspezifikation kann der Ablauf des Protokolls und auch die Kommunikation der Protokolleinheiten untereinander schon vor der Implementation getestet werden. Dadurch können Fehler im logischen Protokollablauf schon nach erfolgter Protokollspezifikation ge-

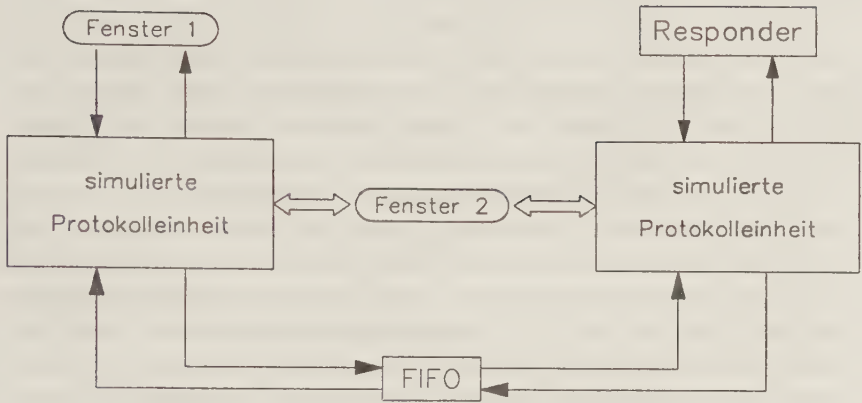


Bild 7.4-3 Simulation mit zwei Protokolleinheiten
und Testresponder

funden werden. Durch die rechnergestützte Erstellung der Listen sind Fehler schnell korrigierbar. Da die Dokumentation vom Rechner alleine erstellt wird, erhält man somit nach Korrekturen auch automatisch eine neue Protokollspezifikation.

8 Erfahrungen und Ausblick

Der praktische Einsatz des Protokollentwurfsverfahrens hat gezeigt, daß eine schnelle Einarbeitung in diese Spezifikationsmethode gegeben ist. Eine Hilfe ist dabei die Menue-Oberfläche des Programmpaketes zur Erstellung der Listen, die den Anwender schrittweise durch die Spezifikation führt. Die so entstandenen Protokollspezifikationen waren gut als Implementationsgrundlage geeignet, so daß die spezifizierten Protokolle leicht auch von anderen Personen implementiert werden konnten und eine gute Einarbeitung in die jeweilige Protokollproblematik gegeben war. Außer den selbst entwickelten Protokollen wurde auch das Token-Passing Bus Protokoll /802.484/ mit dieser Methode spezifiziert /SÖN86/. Dabei mußten jedoch in dieser Normen-Spezifikation vorkommende "Pseudo-Zustände" entfernt werden /HÖL 86/.

Beim Entwurf von Protokollen der Anwenderschicht hat sich gezeigt, daß die Schnittstelle des Protokolls zum Betriebssystem als SAP ausgelegt werden kann /JÄG86/FRE86/. Damit wird eine gewisse Betriebssystemunabhängigkeit erreicht, und die Protokolleinheit kann auf asynchrone Ereignisse des Betriebssystems reagieren.

Die Funktion und der logische Ablauf der jeweiligen Protokolle muß natürlich vom Entwickler festgelegt werden. Da für die entwickelten Protokolle jedoch eine sehr exakte und eindeutige Protokollspezifikation zur Verfügung steht, können in den Protokollen Mechanismen, wie das Weiterleiten der Adresse eines Datenblocks über die SAPs, statt des Umkopierens der Daten von Ebene zu Ebene, zum Einsatz kommen, ohne die Unsicherheit in Kauf nehmen zu müssen, daß die benötigten Daten überschrieben oder gelöscht werden. Um einen schnellen Datentransfer zu ermöglichen, ist es erforderlich, diesen Mechanismus in allen Ebenen einzusetzen, was eine konsequente Protokollplanung über alle Ebenen erfordert.

Bei den bisher mit dieser Methode spezifizierten Protokollen hat sich gezeigt, daß die Anzahl der möglichen Variablen, Konstanten usw. ausreichend war. Es ist jedoch möglich, die Durchnummerierung von zweistelligen Hexadezimalziffern auf z. B. dreistellige Dezimalziffern zu erweitern, um so eine größere Anzahl von Elementen definieren zu können.

Für eine Simulation des Zeitverhaltens des Protokolls können die Listen um Zeiten erweitert werden, die dann für jede Operation eine minimale, eine typische und eine maximale Ausführungszeit angeben. Diese Zeiten hängen jedoch stark von der Realisierung des Protokolls ab, wobei nicht nur ein großer Unterschied zwischen einer Hardwarelösung und einer Softwarelösung besteht, sondern bei einer Realisierung als Programm die gesamte Ausführungszeit auch noch von der momentanen Auslastung des Rechners abhängt.

Inwieweit eine automatische Codegenerierung oder der automatische Entwurf einer Schaltung auf Basis der Protokollspezifikation durchgeführt werden kann, wäre noch zu untersuchen.

9 Anhang

9.1 Struktur der zur Protokollspezifikation und Dokumentation verwendeten Listen.

- 1 Liste der Zustände
- 2 Liste der Konstanten
- 3 Liste der Variablen
- 4 Liste der Felder
- 5 Liste der Common-Felder
- 6 Liste der Datenblöcke
- 7 Liste der unabhängigen Ereignisse
- 8 Liste der Schnittstellenaktionen
- 9 Verknüpfung von Datenblockelementen zu unabhängigen Ereignissen
- 10 Zuweisung von Datenblockelementen bei Schnittstellenaktionen
- 11 Liste der Verknüpfungen nach abhängigen Ereignissen ausgerichtet
- 12 Liste der Verknüpfungen nach unabhängigen Ereignissen ausgerichtet
- 13 Liste der abhängigen Ereignisse
- 14 Liste der Zustandsübergänge
- 15 Liste der zustandsunabhängigen Aktionen
- 16 Liste der zustandsabhängigen Aktionen
- 17 Liste der Aktionen der Funktionen
- 18 Liste der Funktionen
- 19 Definition der Unterzustandsmaschinen

Prinzipiell gilt für alle folgenden Listen:

- xx - bedeutet Hexadezimalzahl von 00 bis FF
- yy - bedeutet Hexadezimalzahl von 00 bis FF
- zz - bedeutet Hexadezimalzahl von 00 bis FF
- n - bedeutet Hexadezimalzahl von 0 bis F
- i - bedeutet Integer Zahl

Sämtlichen Listen geht eine Seite Beschreibung voraus, in der kurz die Struktur der Liste beschrieben wird. Die gesamte Protokollspezifikation ist in der Form

Protokollname Seite uuu / vvv Liste xx Seite yyy / zzz

durchnummeriert.

vvv gibt die gesamte Anzahl der Seiten der Protokoll-
spezifikation an

uuu gibt die aktuelle Seite der gesamten Protokoll-
spezifikation an

xx gibt die Nummer (01 - 19) der gerade aktuellen
Liste an

zzz gibt an wieviele Seiten zur aktuellen Liste gehören

yyy gibt die aktuelle Seite innerhalb der Liste an

1 Liste der Zustände

In dieser Liste werden alle Zustände definiert. Die Zustände werden mit Z_xx bezeichnet.

Die Liste hat folgende Form:

Nr.	Name des Zustandes	Bemerkungen
Z_xx	Zustandsname	erklärender Text
.	..	...
.	..	...

2 Liste der Konstanten

In dieser Liste können Konstanten für die Protokolleinheit definiert werden. Konstanten werden mit K_xx bezeichnet. Es können nur Integer Werte zugewiesen werden.

Nr.	Konstante	Wert	Bemerkungen
K_xx	Konstantenname	i	erklärender Text
.	..	.	...
.	..	.	...

3 Liste der Variablen

Diese Liste enthält die Variablen des Protokolls. Variablen werden mit V_xx bezeichnet. Der Wertebereich der Variablen kann sowohl über Integer Zahlen, als auch über Konstanten festgelegt werden.

Für boolesche Variablen gilt folgende Vereinbarung:

FALSE = 0

TRUE = alle übrigen Werte

Nr.	Name	von	bis	Bemerkungen
V_xx	Variablenname	i	k	erklärender Text
V_xx	Variablenname	i	K_xx	erklärender Text
V_xx	Variablenname	K_xx	i	erklärender Text
V_xx	Variablenname	K_xx	K_yy	erklärender Text
.	..	.	.	...
.	..	.	.	...

4 Liste der Felder

In dieser Liste werden Felder definiert. Die prinzipielle Bezeichnung für ein Feld sieht folgendermaßen aus:

$F_{xx}(V_{yy})$

wobei V_{yy} eine definierte Variable sein muß. Der Wertebereich von V_{yy} gibt dabei die Größe des Feldes an. Der Wertebereich der einzelnen Feldelemente wird gesondert angegeben.

Beim Zugriff auf Feldelemente sind Integer-Zahlen, Konstanten, Variablen und Feldelemente erlaubt. Wird auf ein Feldelement über den Wert eines anderen Feldelementes zugegriffen, so sind bei der Spezifikation des zweiten Feldelementes nur noch Variablen, Konstanten oder Integer zulässig.

Nr.	Name	von	bis	Bemerkungen
$F_{xx}(V_{yy})$	Feldname	i	K_{xx}	erklärender Text
..	..	.	.	...
..	..	.	.	...

5 Liste der Common-Felder

In dieser Liste werden Felder definiert, die global für die gesamte Spezifikation vereinbart werden. Die prinzipielle Bezeichnung für ein Common-Feld sieht folgendermaßen aus:

$C_{xx}(V_{yy})$

wobei V_{yy} eine definierte Variable sein muß. Der Wertebereich von V_{yy} gibt dabei die Größe des Feldes an. Der Wertebereich der einzelnen Feldelemente wird gesondert angegeben.

Beim Zugriff auf Common-Feldelemente sind Integer-Zahlen, Konstanten, Variablen und Feldelemente erlaubt. Wird auf ein Feldelement über den Wert eines anderen Feldelementes zugegriffen, so sind bei der Spezifikation des zweiten Feldelementes nur noch Variablen, Konstanten oder Integer zulässig.

Die Common-Felder werden in der Hauptzustandsmaschine definiert und sind in allen Unterzustandsmaschinen benutzbar.

Nr.	Name	von	bis	Bemerkungen
$C_{xx}(V_{yy})$	Feldname	i	K_{xx}	erklärender Text
..	..	.	.	...
..	..	.	.	...

6 Liste der Datenblöcke

In dieser Liste werden die Datenblöcke bei Ereignissen und für Schnittstellenaktionen definiert. Die Bezeichnung lautet: D_0n(V_xx) bzw. D_1n(V_xx), wobei folgendes gilt:

- D_0n Datenblock am SAP n bei Ereignissen, die auf die Protokolleinheit wirken
- D_1n Datenblock am SAP n bei Schnittstellenaktionen der Protokolleinheit

Der Wertebereich von V_xx gibt die Größe des Datenblocks an, der Wertebereich der einzelnen Datenblockelemente wird gesondert angegeben.

Die Datenblöcke D_0n und D_1n sind in der Hauptzustandsmaschine und in allen Unterzustandsmaschinen identisch.

Nr.	Name	von	bis	Bemerkungen
D_0n(V_xx)	Block-Name	i	K_xx	Anford. an Ebene
..	..	.	.	...
..	..	.	.	...

7 Liste der unabhängigen Ereignisse

In dieser Liste werden die unabhängigen Ereignisse, die auf die Protokolleinheit wirken, definiert. Die Bezeichnung der Ereignisse erfolgt folgendermaßen:

I_xx	Inneres Ereignis (abgelaufener Timer)
An_xx	Anforderung xx von SAP n
Bn_xx	Bestätigung xx von SAP n
Mn_xx	Meldung xx von SAP n
Rn_xx	Antwort (Response) xx von SAP n

Diese Liste wird für jeden SAP neu erstellt.

Nr.	Name des unabh. Ereignisses	Bemerkungen
An_xx	Name der Anforderung	erklärender Text
Bn_xx	Name der Bestätigung	erklärender Text
Mn_xx	Name der Meldung	erklärender Text
Rn_xx	Name der Antwort	erklärender Text
I_xx	Name des Timers	erklärender Text
.	..	...
.	..	...

8 Liste der Schnittstellenaktionen

In dieser Liste werden die Schnittstellenaktionen der Protokolleinheit nach außen definiert. Die Bezeichnung der Aktionen erfolgt folgendermaßen:

An_xx	Anforderung xx an SAP n
Bn_xx	Bestätigung xx an SAP n
Mn_xx	Meldung xx an SAP n
Rn_xx	Antwort (Response) xx an SAP n

Diese Liste wird für jeden SAP neu erstellt.

Nr.	Name der Schnittstellenakt.	Bemerkungen
An_xx	Name der Anforderung	erklärender Text
Bn_xx	Name der Bestätigung	erklärender Text
Mn_xx	Name der Meldung	erklärender Text
Rn_xx	Name der Antwort	erklärender Text
.	..	...
.	..	...

9 Verknüpfung von Datenblockelementen zu unabhängigen Ereignissen

In dieser Liste sind die Verknüpfungen aufgeführt, wie aus Datenblockelementen unabhängige Ereignisse abgeleitet werden. In der ersten Spalte stehen die benötigten Datenblockelemente, in den folgenden Spalten die unabhängigen Ereignisse, die ausgelöst werden, wenn die Verknüpfungen innerhalb der Spalte TRUE ergeben.

In der Liste sind folgende Operatoren zulässig:

==	gleich
/=	ungleich
>	größer als
>=	größer gleich
<	kleiner als
<=	kleiner gleich
+	Addition
-	Subtraktion
**	Multiplikation
//	Division
/	Negation (Zweierkomplement)
&	bitweises logisches AND
/&	bitweises logisches NAND
\$	bitweises logisches OR
/\$	bitweises logisches NOR
<-	Rotation links
->	Rotation rechts
=&	Gleich der Adresse eines Feldes
=*	Gleich dem Inhalt eines Feldes über Adresse

Als Elemente für die Vergleichsoperationen sind Integer-Zahlen, Konstanten, Variablen und Feldelemente zugelassen. Diese Liste wird für jeden SAP neu erstellt.

	Xn_xx	Xn_yy
D_0n(i)	== K_yy	>= V_zz
D_0n(k)	/= 0	== V_yy & K_zz
.	.	.
.	.	.

10 Zuweisung von Datenblockelementen bei Schnittstellenaktionen

In dieser Liste wird angegeben, welche Datenblockelemente mit welchem Wert belegt werden, wenn die entsprechende Schnittstellenaktion ausgelöst wird. In der ersten Spalte stehen die Datenblockelemente, und in den folgenden Spalten die Zuweisungen für die entsprechenden Schnittstellenaktionen.

Diese Liste wird für jeden SAP neu erstellt.

In der Liste sind folgende Operatoren definiert:

- := Wertzuweisung
- + Addition
- Subtraktion
- ** Multiplikation
- // Division
- / Negation (Zweierkomplement)
- & bitweises logisches AND
- /& bitweises logisches NAND
- \$ bitweises logisches OR
- /\$ bitweises logisches NOR
- <- Rotation links
- > Rotation rechts
- =& Zuweisung der Adresse eines Feldes
- =* Zuweisung eines Feldes über Adresse

	Xn_yy	Xn_zz
D_1n(i)	:= K_xx	
D_1n(k)	:= 15	-> 5
D_1n(1)	=* V_xx, K_yy	
.	.	.
.	.	.

11 Liste der Verknüpfungen nach abhängigen Ereignissen ausgerichtet

In dieser Liste erfolgt die Verknüpfung von unabhängigen Ereignissen mit Variablen und Feldelementen zu abhängigen Ereignissen.

In der ersten Spalte der Liste werden alle benötigten Variablen, Feldelemente und unabhängigen Ereignisse aufgeführt. Für die Liste gibt es zwei Alternativen.

Eine Liste, die OR verknüpfte AND Terme enthält
(disjunktive Form)

Eine Liste, die AND verknüpfte OR Terme enthält
(konjunktive Form)

Für jedes abhängige Ereignis wird eine neue Liste erstellt.

Das abhängige Ereignis tritt ein, wenn das Ergebnis der Verknüpfung, die durch die Liste beschrieben ist, den Wert TRUE ergibt.

Als Elemente für die Vergleichsoperationen sind Integer - Zahlen, Konstanten, Variablen und Feldelemente zugelassen.

Über der jeweiligen Liste steht das abhängige Ereignis, dessen Namen und eine Bemerkung.

In der Liste sind folgende Operatoren zulässig:

*	unabhängiges Ereignis ist eingetreten
==	gleich
/=	ungleich
>	größer als
>=	größer gleich
<	kleiner als
<=	kleiner gleich
+	Addition
-	Subtraktion
**	Multiplikation
//	Division
/	Negation (Zweierkomplement)
&	bitweises logisches AND
/&	bitweises logisches NAND
\$	bitweises logisches OR
/\$	bitweises logisches NOR
<-	Rotation links
->	Rotation rechts
=&	Gleich der Adresse eines Feldes
=*	Gleich dem Inhalt eines Feldes über Adresse

E_xx	Ereignisname	erklärender Text
------	--------------	------------------

AND	OR 1
-----	------

Xn_yy	*
V_xx	== K_zz
.	.
.	.

AND	OR 2
-----	------

Xn_yy	*
F_xx(V_yy)	== V_zz /& K_xx
.	.
.	.

Die alternative Form der Liste hat folgendes Aussehen:

E_xx	Ereignisname	erklärender Text
------	--------------	------------------

OR	AND 1
----	-------

Xn_yy	*
V_xx	== K_zz
.	.
.	.

OR	AND 2
----	-------

Xn_yy	*
F_xx(V_yy)	== V_zz /& K_xx
.	.
.	.

12 Liste der Verknüpfungen nach unabhängigen Ereignissen ausgerichtet

In dieser Liste erfolgt die Verknüpfung von unabhängigen Ereignissen mit Variablen und Feldelementen zu abhängigen Ereignissen.

Diese Liste ist inhaltlich mit Liste 11 identisch, nur daß sie nach unabhängigen Ereignissen ausgerichtet ist.

Diese Liste braucht vom Benutzer nicht erstellt zu werden, sondern wird vom Rechner aus Liste 11 generiert.

In der ersten Spalte der Liste werden alle benötigten Variablen, Feldelemente und unabhängige Ereignisse aufgeführt. In den einzelnen Spalten werden die entsprechenden Verknüpfungen angegeben, die erfüllt sein müssen, damit das abhängige Ereignis generiert wird.

Das abhängige Ereignis tritt ein, wenn das Ergebnis der Verknüpfung innerhalb der Spalte den Wert TRUE ergibt.

Für jedes unabhängige Ereignis wird eine neue Liste erstellt.

Als Elemente für die Vergleichsoperationen sind Integer-Zahlen, Konstanten, Variablen und Feldelemente zugelassen.

Über der jeweiligen Liste steht das unabhängige Ereignis.

In der Liste sind folgende Operatoren zulässig:

*	unabhängiges Ereignis ist eingetreten
==	gleich
/=	ungleich
>	größer als
>=	größer gleich
<	kleiner als
<=	kleiner gleich
+	Addition
-	Subtraktion
**	Multiplikation
//	Division
/	Negation (Zweierkomplement)
&	bitweises logisches AND
/&	bitweises logisches NAND
\$	bitweises logisches OR
/\$	bitweises logisches NOR
<-	Rotation links
->	Rotation rechts
=&	Gleich der Adresse eines Feldes
=*	Gleich dem Inhalt eines Feldes über Adresse

X_xx	Ereignisname	erklärender Text
------	--------------	------------------

	E_xx	E_yy
--	------	------

V_xx	== K_zz	
------	---------	--

V_yy	. .	> V_zz
------	-----	--------

.	.	
---	---	--

.	.	
---	---	--

13 Liste der abhängigen Ereignisse

Diese Liste braucht vom Anwender nicht erstellt zu werden. Sie wird aus Liste 11 vom Rechner generiert. Die Bezeichnung für die abhängigen Ereignisse lautet E_xx.

Nr.	abhäng. Ereignis	Bemerkung
E_xx	Ereignisname	erklärender Text
.	..	...
.	..	...

14 Liste der Zustandsübergänge

In dieser Liste werden die Zustandsübergänge der Protokoll-einheit festgelegt. Die Liste enthält jeweils den aktuellen Zustand, das abhängige Ereignis, das einen Zustandsübergang bewirkt, und den Folgezustand.

Aktueller Zust.	Ereignis	Folgezustand
Z_xx Name	E_yy Name	Z_zz Name
. ..	. ..	. ..
. ..	. ..	. ..

15 Liste der zustandsunabhängigen Aktionen

In dieser Liste werden die Aktionen definiert, die beim Eintreten eines unabhängigen Ereignisses ausgelöst werden können.

In der ersten Spalte der Liste stehen die zu modifizierenden Parameter und Schnittstellenaktionen. In den folgenden Spalten jeweils die durchzuführenden Operationen für das entsprechende unabhängige Ereignis, wobei das Auslösen einer Schnittstellenaktion immer die letzte ausgeführte Operation ist.

Reichen die Spalten nicht aus, so wird eine Liste mit den folgenden Spalten angehängt.

In der Liste sind folgende Operatoren definiert:

- * Schnittstellenaktion auslösen
- *n wie *, n gibt die Reihenfolge der Aktionen an
- := Wertzuweisung
- + Addition
- Subtraktion
- ** Multiplikation
- // Division
- / Negation (Zweierkomplement)
- & bitweises logisches AND
- /& bitweises logisches NAND
- \$ bitweises logisches OR
- /\$ bitweises logisches NOR
- <- Rotation links
- > Rotation rechts
- =& Zuweisung der Adresse eines Feldes
- =* Zuweisung eines Feldes über Adresse

	Xn_xx	Xn_yy
Xn_yy	*	
V_xx	:= 15	-> 5
F_xx (F_yy (V_zz)	=* V_xx, K_yy	
Xn_zz		*
.	.	.
.	.	.

16 Liste der zustandsabhängigen Aktionen

In dieser Liste werden die Aktionen definiert, die beim Eintreten eines zustandsabhängigen Ereignisses ausgelöst werden.

In der ersten Spalte der Liste stehen die zu modifizierenden Parameter und Schnittstellenaktionen. In den folgenden Spalten jeweils die durchzuführenden Operationen für das entsprechende abhängige Ereignis, wobei das Auslösen einer Schnittstellenaktion immer die letzte ausgeführte Operation ist.

In der Liste sind folgende Operatoren definiert:

- * Schnittstellenaktion auslösen
- *n wie *, n gibt die Reihenfolge der Aktionen an
- := Wertzuweisung
- + Addition
- Subtraktion
- ** Multiplikation
- // Division
- / Negation (Zweierkomplement)
- & bitweises logisches AND
- /& bitweises logisches NAND
- \$ bitweises logisches OR
- /\$ bitweises logisches NOR
- <- Rotation links
- > Rotation rechts
- =& Zuweisung der Adresse eines Feldes
- =* Zuweisung eines Feldes über Adresse
- *+ Unterzustandsmaschine kreieren
- *- Unterzustandsmaschine löschen
- >> Unabhängiges Ereignis von SAP n an UZM weiterleiten

Für jeden Zustand wird eine eigene Liste angelegt.

Reichen die Spalten nicht aus, so wird eine Liste mit den folgenden Spalten angehängt.

Z_XX Offline

	E_xx	E_yy
Xn_yy	*	
V_xx	:= 15	-> 5
F_xx(F_yy(V_zz)	=* V_xx,K_yy	
Xn_zz		*
.	.	.
.	.	.

17 Liste der Aktionen der Funktionen

In dieser Liste werden die Funktionen definiert, die von den zustandsabhängigen Aktionen der Protokolleinheit aufrufbar sind. Bezeichnet wird eine Funktion in der Form $P_{xx}(V_{yy})$, wobei V_{yy} den Wertebereich des von der Funktion zurückgelieferten Parameters spezifiziert.

In der ersten Spalte der Liste stehen die von der Funktion zu modifizierenden Parameter und Schnittstellenaktionen. In den folgenden Spalten jeweils die nacheinander auszuführenden Aktionen, wobei das Auslösen einer Schnittstellenaktion immer die letzte ausgeführte Operation ist.

In der Liste sind folgende Operatoren definiert:

- * Schnittstellenaktion auslösen
- *n wie *, n gibt die Reihenfolge der Aktionen an
- := Wertzuweisung
- + Addition
- Subtraktion
- ** Multiplikation
- // Division
- / Negation (Zweierkomplement)
- & bitweises logisches AND
- /& bitweises logisches NAND
- \$ bitweises logisches OR
- /\$ bitweises logisches NOR
- <- Rotation links
- > Rotation rechts
- =& Zuweisung der Adresse eines Feldes
- =* Zuweisung eines Feldes über Adresse
- *+ Unterzustandsmaschine kreieren
- *- Unterzustandsmaschine löschen
- >> Unabhängiges Ereignis von SAP n an UZM weiterleiten

Reichen die Spalten nicht aus, so wird eine Liste mit den folgenden Spalten angehängt.

P_xx(V_yy) Funktionsname erklärender Text

Aktion 1

Aktion 2

Xn_yy

*

V_xx

:= 15

-> 5

F_xx(F_yy(V_zz)

=* V_xx,K_yy

Xn_zz

*

.

. .

. .

.

. .

. .

18 Liste der Funktionen

Diese Liste braucht vom Anwender nicht erstellt zu werden, sie wird aus Liste 17 vom Rechner generiert. Bezeichnet werden Funktionen mit P_xx

Nr.	Name der Funktion	Bemerkungen
P_xx(V_yy)	Funktionsname	erklärender Text
.	..	...
.	..	...

19 Definition der Unterzustandsmaschinen

Die Definition einer Unterzustandsmaschine (UZM) besteht aus den gleichen Listen (01 - 19). Die Definition hat folgendes Aussehen: U_0n(yy).

n bezeichnet die Nummer der UZM von 1 bis F, und yy wieviele UZM dieses Typs kreiert werden können. In der Definition wird der Dateinamen angegeben, wo die Beschreibung abgelegt ist. Diese Beschreibung wird getrennt nach dem gleichen Verfahren erstellt.

Parallel zu jeder UZM-Definition gibt es noch ein Feld U_1n mit yy Elementen, wobei jedes Element dieses Feldes einer UZM dieses Typs zugeordnet ist. In diesen Elementen ist vermerkt (TRUE oder FALSE), ob die entsprechende UZM existiert oder nicht. Die Unterzustandsmaschine selbst kann über U_00 auf ihr entsprechendes Feldelement in der übergeordneten Zustandsmaschine zugreifen. Die UZM sind von 1 - yy durchnummeriert. U_1n(00) gibt dabei immer die nächste freie noch nicht kreierte bzw. wieder gelöschte UZM an.

Das Feld U_1n in der übergeordneten Zustandsmaschine, und die Variable U_00 in der Unterzustandsmaschine existieren immer parallel zu einer UZM-Definition, und müssen nicht explizit beschrieben werden.

Die Datenblöcke D_0n und D_1n sowie die Common-Felder sind in der Hauptzustandsmaschine und in allen Unterzustandsmaschinen identisch.

Nr.	Dateiname	Bemerkungen
U_0n(K_xx)	Name	erklärender Text
U_0n(i)	Name	erklärender Text
..	..	...
..	..	...

9.2 Literaturverzeichnis

- /ANS82/ J.P. Ansart, O. Rafiq, V. Chari

PDIL Protocol Description and Implementation
Language

Protocol Specification, Testing, and Verification
S. 101 - 112

North Holland Publishing Company
Amsterdam, New York, Oxford 1982
ISBN 0-444-86481-4
- /BEH86/ Hans-Heinrich Behrmann

Entwicklung einer universellen Protokoll-Simulation
auf Basis der "erweiterten Zustandsdiagramm-
technik"

Diplomarbeit D 150 am Lehrstuhl für Datenverarbeitung
Bochum 1986
- /BIL83/ J. Billington

Abstract Specification of The ISO Transport
Service Definition using Labelled Numerical
Petri-Nets

Protocol Specification, Testing, and Verification
III S. 173 - 185

North Holland Publishing Company
Amsterdam, New York, Oxford 1983
ISBN 0-444-86769-4

- /BOC86/ Peter Bocker
- ISDN - Das dienstintegrierende digitale
Nachrichtennetz
Konzept, Verfahren, Systeme
- Springer-Verlag
Berlin, Heidelberg, New York, Tokyo 1986
ISBN 3-540-15727-1
- /DAT83/ DATEX-P-Handbuch
- Bundesdruckerei
Fernmeldetechnisches Zentralamt
Darmstadt 1983
- /EH483/ DuD Fachbeiträge 2
- Einheitliche höhere Kommunikationsprotokolle
Schicht 4
- Vieweg Verlag
Braunschweig 1983
ISBN 3-528-03593-5
- /EH583/ DuD Fachbeiträge 3
- Einheitliche höhere Kommunikationsprotokolle
Schichten 5 und 6
- Vieweg Verlag
Braunschweig 1983
ISBN 3-528-03595-1

- /EXE82/ M. Exel et al.
SL1 Language
A Specification and Design Tool for Switching
Systems Software Development
IEEE Transactions on Communications
Communications Software
Vol. COM-30 No. 6, June 1982
S. 92 - 98
Prentice-Hall, Inc.
Englewood Cliffs, N.J. 1983
ISBN 0-13-450271-X
- /FRE86/ Matthias Frei
Entwicklung einer betriebssystemunabhängigen Pro-
tokolleinheit für Dateitransfer in einem offenen
Mikrorechnernetz
Diplomarbeit D 138 am Lehrstuhl für Datenverar-
beitung
Bochum 1986
- /GEW85/ Gewalt, Haake, Pfadler
Software Engineering
Grundlagen und Technik rationeller Programment-
wicklung
R. Oldenbourg Verlag
München Wien 1985
ISBN 3-486-21494-2

- /HÖL86/ R. Hölzgen, R. König, H. Thomalla, W. Weber
Computer Networking - Experiences with Protocol
Descriptions by State Diagrams
XXX Jugoslawische Konferenz ETAN
Herceg-Novi 1986
- /HÖL87/ R. Hölzgen, W. Weber
Ein Entwurfsverfahren für Protokolle in Rechner-
netzen
4. Fachtagung "Beiträge für eine zukunftsweisende
Robotertechnik" Dezember 1987
Wuppertal 1987
- /JÄG86/ Hermann Jäger
Entwicklung eines betriebssystemunabhängigen
Message-Handling-Systems für ein offenes lokales
Mikrorechnernetz
Diplomarbeit D 151 am Lehrstuhl für Datenver-
arbeitung
Bochum 1986
- /PAR72/ D. Parnas
On the criteria to be user in decomposing systems
into Modules
CACM 15/72 S. 1053 - 1058

- /PIA86/ T. F. Piatkowski, R. C. Venkatraman
A formal comparsion of formal protocol
specification techniques

Protocol Specification, Testing, and Verification
V S. 401 - 420

North Holland Publishing Company
Amsterdam, New York, Oxford 1986
ISBN 0-444-87881-5
- /RAD88/ Friedrich Rademacher

Entwurf, Spezifikation und Implementation eines
Transportprotokolls für einen modularen Front-
End-Rechner

Diplomarbeit D 164 am Lehrstuhl für Datenverar-
beitung
Bochum 1988
- /REK82/ Kirsten Rekdal

CHILL - The Standard Language for Programming SPC
Systems

IEEE Transactions on Communications
Communications Software
Vol. COM-30 No. 6, June 1982
S. 54 - 64

Prentice-Hall, Inc.
Englewood Cliffs, N.J. 1983
ISBN 0-13-450271-X

- /SCH82/ Peter Schnupp
- Rechnernetze
Entwurf und Realisierung
- de Gruyter
New York, Berlin 1982
ISBN 3-11-008951-3
- /SCI83/ Schicker P.
- Datenübertragung und Rechnernetze
- Teubner
Stuttgart 1983
ISBN 3-519-02463-2
- /SIM82/ G. A. Simon, D. J. Kaufman
- An extended finite state machine approach to
protocol specification
- Protocol Specifiation, Testing, and Verification
S. 113 - 133
- North Holland Publishing Company
Amsterdam, New York, Oxford 1982
ISBN 0-444-86481-4
- /SNE79/ P. Schnupp, C. Floyd
- Software
Programmentwicklung und Projektorganisation
- de Gruyter
New York, Berlin 1979
ISBN 3-11-007865-1

/SÖN86/ Volker Söntgerath

Entwicklung einer universellen Protokoll-Dokumentation auf Basis der "erweiterten Zustandsdiagrammtechnik"

Diplomarbeit D 143 am Lehrstuhl für Datenverarbeitung
Bochum 1986

/SUN81/ Carl A. Sunshine

Formal Modeling of Communication Protocols

Informatik-Fachberichte, Band 40
Kommunikation in verteilten Systemen
S. 406 - 428

Springer Verlag
Berlin Heidelberg New York 1981
ISBN 3-540-10618-9

/TAN81/ A. S. Tanenbaum

Computer Networks

Prenzice Hall
Englewood Cliffs N.J. 1981
ISBN 0-13-164699-0

/TEG88/ Matthias Tegethoff

Erstellung eines Programmpaketes zur interaktiven Eingabe von Listen zur Protokollspezifikation

Studienarbeit S 140 am Lehrstuhl für Datenverarbeitung
Bochum 1988

/THO85/ Hartmut Thomalla

Ein modularer Front-End-Rechner mit Knotenfunk-
tion für ein offenes lokales Mikrorechnernetz

VDI-Verlag
Düsseldorf 1985
ISBN 3-18-144710-2

/TIE82/ W. Tietz (Hrsg)

CCITT Empfehlungen der V-Serie und der X-Serie
Band 3 Datenübermittlungsnetze
S. 38 - 100

R. v. Decker's Verlag
Heidelberg Hamburg 1982
ISBN 3-7685-2982-7

/VAL85/ Isabelle Valet, Dominique Rerat

Formal Description of the NADIR Transport
Protocol using the PDIL Language

Protocol Specification, Testing, and Verification
IV S. 147 - 163

North Holland Publishing Company
Amsterdam, New York, Oxford 1985
ISBN 0-444-87676-6

/VER87/ Martin Verlande

Entwicklung und Implementierung eines Token-
passing Busprotokolls auf ArcNet-Basis für einen
modularen Front-End-Rechner

Diplomarbeit D 153 am Lehrstuhl für Datenver-
arbeitung
Bochum 1987

/WAL83/ Bernd Walter

Timed Petri-Nets for Modeling and Analyzing
Protocols with Real-Time Characteristics

Protocol Specification, Testing, and Verification
III S. 149 - 159

North Holland Publishing Company
Amsterdam, New York, Oxford 1983
ISBN 0-444-86769-4

/WEB84/ Wolfgang Weber

Rechnernetze

Scriptum zur Vorlesung im SS 1984
Ruhr Universität Bochum

/YEL81a/ Yellow Book Volume VIII Fascicle VIII.2

Data Communication Networks Services and
Facilities

Terminal Equipment and Interfaces

Recommendations X.1 - X.29

CCITT

Genf 1981

ISBN 92-61-01161-6

/YEL81b/ Yellow Book Volume VI Fascicle VI.7

Functional Specification and Description Language
(SDL)

Recommendations Z.101 - Z.104

CCITT

Genf 1981

ISBN 92-61-01111-X

/ZIM81/ H. Zimmerman

The ISO Reference Model for Open Systems
Interconnection

Informatik-Fachberichte Band 40
Kommunikation in verteilten Systemen
S. 39 - 57

Springer Verlag
Berlin 1981
ISBN 3-540-10618-9

/ZUS80/ Konrad Zuse

Petri-Netze aus der Sicht des Ingenieurs

Vieweg Verlag
Braunschweig 1980
ISBN 3-528-09615-2

/802.385/ IEEE Standards for Local Area Networks

Carrier Sense Multiple Access with
Collision Detection

Access Method and Physical Layer Specification
ANSI/IEEE Std. 802.3-1985

IEEE
New York 1985
ISBN 0-471-82749-5

/802.485/ IEEE Standards for Local Area Networks
Token Passing Bus Access Method and
Physical Layer Specifications
ANSI/IEEE 802.4-1985
ISO Draft International Standard 8802/4

IEEE
New York 1985
ISBN 0-471-82750-9

/802.585/ IEEE Standards for Local Area Networks
Token Ring Access Method and
Physical Layer Specifications
ANSI/IEEE Std. 802.5-1985
ISO Draft Proposal 8802/5

IEEE
New York 1985
ISBN 0-471-82996-X

Lebenslauf

06.01.1958	geboren in Dernbach/Westerwald
1964-1967	Grundschulausbildung in der Volksschule Staudt
1967-1976	Besuch des staatlichen Gymnasiums Montabaur
1976-1977	Grundwehrdienst
1977-1984	Studium der Elektrotechnik an der Ruhr-Universität Bochum mit Diplomarbeit am Lehrstuhl für Datenverarbeitung von Herrn Prof. Dr.-Ing. W. Weber
seit 1984	wissenschaftlicher Mitarbeiter an diesem Lehrstuhl

